(* APPENDIX *)

(* IMPLEMENTATION for IBM PERSONAL COMPUTER of

Prof. N. WIRTH'S PASCAL-S modified from:

1. "Pascal-S: A Subset and its Implementation", N. Wirth, from "PASCAL The Language and its Implementation", edited by D.W. Barron, John Wiley & Sons, 1981, Pp. 199-259; and

2. "Programming Language Translation", R.E. Berry, Ellis Horwood Limited, 1982.

The following source code is in Turbo Pascal by Borland International of Scotts Valley, California. *)

{ The following included files are listed below this main file: }

```
  {$I e:decl3 }
  {$I e:edit4 }
  {$I e:err3 }
  {$I e:lex4 }
  {$I e:entr }
  {$I e:block3 }
  {$I e:intr }

  PROCEDURE compinit ;

    BEGIN
      key[1] := 'and       ' ;
      key[2] := 'array     ' ;
      key[3] := 'begin     ' ;
      key[4] := 'case      ' ;
      key[5] := 'const     ' ;
      key[6] := 'div       ' ;
      key[7] := 'do        ' ;
      key[8] := 'downto    ' ;
      key[9] := 'else      ' ;
      key[10] := 'end       ' ;
      key[11] := 'for       ' ;
      key[12] := 'function  ' ;
      key[13] := 'if        ' ;
      key[14] := 'mod       ' ;
      key[15] := 'not       ' ;
      key[16] := 'of        ' ;
```

```
key[17] := 'or        ' ;
key[18] := 'procedure ' ;
key[19] := 'program   ' ;
key[20] := 'record    ' ;
key[21] := 'repeat    ' ;
key[22] := 'then      ' ;
key[23] := 'to        ' ;
key[24] := 'type      ' ;
key[25] := 'until     ' ;
key[26] := 'var       ' ;
key[27] := 'while     ' ;
ksy[1] := andsy ;
ksy[2] := arraysy ;
ksy[3] := beginsy ;
ksy[4] := casesy ;
ksy[5] := constsy ;
ksy[6] := idiv ;
ksy[7] := dosy ;
ksy[8] := downtosy ;
ksy[9] := elsesy ;
ksy[10] := endsy ;
ksy[11] := forsy ;
ksy[12] := functionsy ;
ksy[13] := ifsy ;
ksy[14] := imod ;
ksy[15] := notsy ;
ksy[16] := ofsy ;
ksy[17] := orsy ;
ksy[18] := proceduresy ;
ksy[19] := programsy ;
ksy[20] := recordsy ;
ksy[21] := repeatsy ;
ksy[22] := thensy ;
ksy[23] := tosy ;
ksy[24] := typesy ;
ksy[25] := untilsy ;
ksy[26] := varsy ;
ksy[27] := whilesy ;
sps['+'] := plus ;
sps['-'] := minus ;
sps['*'] := times ;
sps['/'] := rdiv ;
sps['('] := lparent ;
sps[')'] := rparent ;
sps['='] := egl ;
sps[','] := comma ;
sps['['] := lbrack ;
sps[']'] := rbrack ;
sps['#'] := neg ;
sps['&'] := andsy ;
sps[';'] := semicolon ;
constbegsys := [plus,minus,intcon,realcon,charcon,ident] ;
typebegsys := [ident,arraysy,recordsy] ;
blockbegsys := [constsy,typesy,varsy,proceduresy,
                  functionsy,beginsy] ;
facbegsys := [intcon,realcon,charcon,ident,lparent,notsy] ;
statbegsys := [beginsy,ifsy,whilesy,repeatsy,forsy,casesy] ;
stantyps := [notyp,ints,reals,bools,chars] ;
lc := 0 ;
ll := 0 ;
```

```
    cc := 0 ;
    ch := ' ' ;
    errpos := 0 ;
    errs := [] ;
    compline := 1 ;
    t := - 1 ;
    a := 0 ;
    b := 1 ;
    sx := 0 ;
    c2 := 0 ;
    display[0] := 1 ;
    iflag := false ;
    oflag := false ;
    enter('          ',variable,notyp,0) ;  (*sentinel*)
    enter('false     ',konstant,bools,0) ;
    enter('true      ',konstant,bools,1) ;
    enter('real      ',typel,reals,1) ;
    enter('char      ',typel,chars,1) ;
    enter('boolean   ',typel,bools,1) ;
    enter('integer   ',typel,ints,1) ;
    enter('abs       ',funktion,reals,0) ;
    enter('sqr       ',funktion,reals,2) ;
    enter('odd       ',funktion,bools,4) ;
    enter('chr       ',funktion,chars,5) ;
    enter('ord       ',funktion,ints,6) ;
    enter('succ      ',funktion,chars,7) ;
    enter('pred      ',funktion,chars,8) ;
    enter('round     ',funktion,ints,9) ;
    enter('trunc     ',funktion,ints,10) ;
    enter('sin       ',funktion,reals,11) ;
    enter('cos       ',funktion,reals,12) ;
    enter('exp       ',funktion,reals,13) ;
    enter('ln        ',funktion,reals,14) ;
    enter('sqrt      ',funktion,reals,15) ;
    enter('arctan    ',funktion,reals,16) ;
    enter('eof       ',funktion,bools,17) ;
    enter('eoln      ',funktion,bools,18) ;
    enter('read      ',prozedure,notyp,1) ;
    enter('readln    ',prozedure,notyp,2) ;
    enter('write     ',prozedure,notyp,3) ;
    enter('writeln   ',prozedure,notyp,4) ;
    enter('          ',prozedure,notyp,0) ;
    WITH btab[1] DO
      BEGIN
        last := t ;
        lastpar := 1 ;
        psize := 0 ;
        vsize := 0
      END ;
    errormsg ;
  END ;
                                          (* compinit  *)

PROCEDURE reinit ;

  BEGIN
    lc := 0 ;
    ll := 0 ;
    cc := 0 ;
    ch := ' ' ;
```

```
      errpos := 0 ;
      errs := [] ;
      compline := 0 ;
      recompile := false ;
      errorstate := false ;
      linebuf := '' ;
    END ;
                                            (*  reinit   *)

  BEGIN                                     (*  main   *)
    filesrch ;
    edinit ;
    pauseline := 0 ;
    IF newfile THEN
      FOR j := 1 TO linelimit DO
        bufarray[j] := NIL ;
(*
       gotoxy( 1, 25);  write('LINENUM = ');
       gotoxy(20, 25);  write('COMPLINE = ');
       gotoxy(40, 25);  write('PAUSELINE = ');
       gotoxy(60, 25);  write('CH = ');
       gotoxy(70, 25);  write('CC = ');
  *)
    gotoxy(col,row) ;
recomp:
    db('START RECOMP:') ;
    compinit ;
    db('AFTER COMPINIT') ;
    reinit ;
    db('AFTER REINIT') ;
    (*  status;  *)
    insymbol ;
    IF sy <> programsy THEN
      error(3)
    ELSE
      BEGIN
        insymbol ;
        IF sy <> ident THEN
          error(2)
        ELSE
          BEGIN
            progname := id ;
            insymbol ;
          END
      END ;
    block(blockbegsys + statbegsys,false,1) ;
    IF recompile THEN
      BEGIN
        recompile := false ;
        GOTO recomp ;
      END ;
    IF sy <> period THEN
      error(22) ;
    emit(31) ;                              (*halt*)
    IF btab[2].vsize > stacksize THEN
      error(49) ;
    IF progname = 'test0     ' THEN
      printtables ;
    IF errs = [] THEN
      interpret
    ELSE
      errormsg ;
    readln ;
  END.                                          (*  main  *)


(******************************************************************)


                          (*  DECL3.PAS  *)

LABEL
  recomp ;
CONST
  nkw = 27 ;    (*no. of key words*)
  alng = 10 ;   (*no. of significant chars in identifiers*)
  llng = 120 ; (*input line length*)
  emax = 322 ; (*max exponent of real numbers*)
  emin = -292 ;(*min exponent*)
  kmax = 15 ;   (*max no. of significant digits*)
  tmax = 100 ; (*size of table*)
  bmax = 20 ;   (*size of block-table*)
  amax = 30 ;   (*size of array-table*)
  c2max = 20 ; (*size of real constant table*)
  csmax = 30 ; (*max no. of cases*)
  cmax = 850 ; (*size of code*)
  lmax = 7 ;    (*maximum level*)
  smax = 600 ; (*size of string-table*)
  ermax = 58 ; (*max error no.*)
  omax = 63 ;   (*highest order code*)
  xmax = 32000; (*2**17 - 1*)
  nmax = 32000; (*2**48-1*)
  lineleng = 136;  (*output line length*)
  linelimit = 3000 ;
  stacksize = 1500 ;

TYPE
  symbol =
(intcon,realcon,charcon,mstring,notsy,plus,minus,times,idiv,rdiv,
imod,andsy,orsy,egl,neg,gtr,geg,lss,leg,lparent,rparent,lbrack,
rbrack,comma,semicolon,period,colon,becomes,constsy,typesy,varsy,
functionsy,proceduresy,arraysy,recordsy,programsy,ident,beginsy,
ifsy,casesy,repeatsy,whilesy,forsy,endsy,elsesy,untilsy,ofsy,dosy,
tosy,downtosy,thensy) ;
  index = - xmax.. + xmax ;
  alfa = STRING[alng] ;
  object = (konstant,variable,type1,prozedure,funktion) ;
  types = (notyp,ints,reals,bools,chars,arrays,records) ;
  symset = SET OF symbol ;
  typset = SET OF types ;
  item = RECORD
           typ : types ;
           ref : index ;
         END ;
  order = PACKED RECORD
                   f : - omax.. + omax ;
                   x : - lmax.. + lmax ;
                   y : - nmax.. + nmax ;
                 END ;
  regrec = RECORD
             ax,bx,cx,dx,bp,si,di,ds,es,flags : integer ;
           END ;
  recptr = ^linerec ;
  linerec = RECORD
              code : STRING[16] ;
              next : recptr ;
            END ;
  loctype = STRING[32] ;
VAR
  sy : symbol; (*last symbol read by insymbol*)
  id : alfa ;  (*identifier from insymbol*)
  inum : integer ; (*integer from insymbol*)
  rnum : real ;    (*real number from insymbol*)
  sleng : integer ;(*string length*)
  ch : char ;      (*last character read from source program*)
  lline : ARRAY [1..llng] OF char ;
  cc : integer ;   (*character counter*)
  lc : integer ;   (*program location counter*)
  ll : integer ;   (*length of current line*)
  errs : SET OF 0..ermax ;
  errpos : integer ;
  progname : alfa ;
  iflag,oflag : boolean ;
  constbegsys,typebegsys,blockbegsys,facbegsys,statbegsys: symset;
  key : ARRAY [1..nkw] OF alfa ;
  ksy : ARRAY [1..nkw] OF symbol ;
  sps : ARRAY [char] OF symbol ;            (*special symbols*)
  t,a,b,sx,c1,c2 : integer ;                (*indices to tables*)
  stantyps : typset ;
  display : ARRAY [0..lmax] OF integer ;
  tab : ARRAY [0..tmax] OF                  (*identifier table*)
  PACKED RECORD
           name : alfa ;
           link : index ;
           obj : object ;
           typ : types ;
           ref : index ;
           normal : boolean ;
           lev : 0..lmax ;
           adr : integer ;
         END ;
  atab : ARRAY [1..amax] OF                 (*array-table*)
  PACKED RECORD
           inxtyp,eltyp : types ;
           elref,low,high,elsize,size : index ;
         END ;
  btab : ARRAY [1..bmax] OF                 (*block-table*)
  PACKED RECORD
           last,lastpar,psize,vsize : index
         END ;
  stab : PACKED ARRAY [0..smax] OF char ;   (*string table*)
  rconst : ARRAY [1..c2max] OF real ;
  code : ARRAY [0..cmax] OF order ;
  psin,psout,prr,prd : text ;
  inf,outf : STRING [24] ;
  i,j : integer ;
  bufarray : ARRAY [1..linelimit] OF recptr ;
```

```
                (*  buffer array of line ptrs  *)
  linenum,topline,lastline,compline,pauseline,k : integer ;
  linebuf : STRING [80] ;
  regs : regrec ;
  row,col : integer ;
  buffed : boolean ;
  inserton : boolean ;
  recompile : boolean ;
  initialized : boolean ;
  c : char ;
  msg : ARRAY [0..ermax] OF alfa ;
  errorstate : boolean ;
  newfile : boolean ;
PROCEDURE debug ;

  BEGIN
    i := i + 1 ;
    gotoxy(0,i) ;
    writeln ;
    write('   cc = ',cc) ;
    write('   ll = ',ll) ;
    write('   ch = ',ch) ;
    writeln ;
  END ;

  PROCEDURE displaysy ;

    BEGIN
      CASE sy OF
        semicolon : BEGIN
                      writeln(lst,' semicolon  ') ; (* readln; *)
                    END ;
        ident : BEGIN
                  writeln(lst,' ident  ') ;  (*  readln;  *)
                END ;
        rparent : BEGIN
                    writeln(lst,' rparent  ') ;  (*  readln;  *)
                  END ;
        varsy : BEGIN
                  writeln(lst,' varsy  ') ;  (*  readln;  *)
                END ;
        forsy : BEGIN
                  writeln(lst,' forsy  ') ;  (*  readln;  *)
                END ;
        dosy : BEGIN
                 writeln(lst,' dosy  ') ;   (*  readln;  *)
               END ;
        becomes : BEGIN
                    writeln(lst,' becomes  ') ;  (*  readln;  *)
                  END ;
        tosy : BEGIN
                 writeln(lst,' tosy  ') ;   (*  readln;  *)
               END ;
        intcon : BEGIN
                   writeln(lst,' intcon  ') ;  (*  readln;  *)
                 END ;
        whilesy : BEGIN
                    writeln(lst,' whilesy  ') ;  (*  readln;  *)
                  END ;
        beginsy : BEGIN
```

```
                    writeln(lst,' beginsy  ') ;  (*  readln;  *)
                  END ;
          ELSE
            BEGIN
              writeln(lst,' unknown  ') ;     (*  readln;  *)
            END ;
        END ;
    END ;

  PROCEDURE tr(i:integer) ;
    (*  trace - writes line number of source code at execution   *)

    BEGIN
(*  gotoxy(1, 16 + i);
  writeln('TRACE LINE ', i);
  gotoxy(col, row);  *)
    END ;

  PROCEDURE status ;
    (*  write status line values of
                linenum, compline, pauseline, cc, ch  *)

    BEGIN
      gotoxy(11,25) ;
      write(linenum) ;
      gotoxy(31,25) ;
      write(compline) ;
      gotoxy(52,25) ;
      write(pauseline) ;
      gotoxy(65,25) ;
      write(ch) ;
      gotoxy(75,25) ;
      write(cc) ;
    END ;

  PROCEDURE db(loc:loctype) ;

    BEGIN
(*
  writeln(lst);
  writeln(lst, loc);
  writeln(lst, 'linenum = ',linenum, '  ', 'topline = ', topline);
  writeln(lst, 'lastline = ', lastline, '  ',
                'compline = ', compline);
  writeln(lst, 'pauseline = ', pauseline, '  ', 'cc = ', cc, '  ',
                'ch = ', ch);
  writeln(lst, linebuf);
  writeln(lst, 'row = ', row, '  ', 'col = ', col);
  if buffed then writeln(lst, 'BUFFED')
                else writeln(lst, 'NOT buffed');
  if inserton then writeln(lst, 'INSERTON')
                else writeln(lst, 'NOT inserton');
  if recompile then writeln(lst, 'RECOMPILE')
                else writeln(lst, 'NOT recompile');
  if initialized then writeln(lst, 'INITIALIZED')
                else writeln(lst, 'NOT initialized');
  if errorstate then writeln(lst, 'ERRORSTATE')
                else writeln(lst, 'NOT errorstate');
  displaysy;
*)
```

```
    END ;


(*****************************************************************)


                      (*  EDIT4.PAS  *)

PROCEDURE clearscreen(rowh,rowl:integer) ;

  BEGIN
    regs.ax := 6 * 256 ; (*  ah = 6;  al = 0  *)
    regs.cx := (rowh - 1) * 256 ; (*  first row cleared  *)
    regs.dx := (rowl - 1) * 256 + 80 ; (* last row cleared  *)
    regs.bx := 7 * 256 ; (*  bh = 7 for black/white attribute  *)
    intr($10,regs) ;
  END ;

  PROCEDURE outch(ch:char) ;
    (*  output char to screen  *)

    BEGIN
      regs.ax := $0A00 + ord(ch) ; (*  AH = 10;  AL = char  *)
      regs.bx := 1 ;
      regs.cx := 1 ;
      intr($10,regs) ;
    END ;

  FUNCTION keyhit : boolean ;
    (*  poll whether key struck  *)

    BEGIN
      regs.ax := 11 * 256 ;                (*  AH = 11  *)
      intr($21,regs) ;
      IF regs.ax > 11 * 256 THEN
        keyhit := true
      ELSE
        keyhit := false ;
    END ;

  FUNCTION inkey : char ;
    (*  returns char if key struck;  otherwise null  *)

    BEGIN
      regs.ax := $600 ;
      regs.dx := $FF ;
      intr($21,regs) ;
      inkey := chr(regs.ax - $600) ;
    END ;
      (*  editor buffer scheme -  array of pointers to linked
          lists of line segments - each segment a record
          linked to the next segment of the same line -
          each record comprises a 16-bit character string
          and a pointer to the next record of same line  *)
```

```
FUNCTION buftolist : recptr ;
      (*  converts 80-byte string (linebuf) to
          linked list of 16-byte records  *)

  VAR
    lptr,rptr,oldptr : recptr ;
    i,numrecs,tail : integer ;

  BEGIN                                    (*  buftolist  *)
    db('START BUFTOLIST') ;
    new(rptr) ;
    rptr^.code := '' ;
    rptr^.next := NIL ;
    lptr := rptr ;
    numrecs := length(linebuf) DIV 16 ;
    tail := length(linebuf) MOD 16 ;
    FOR i := 1 TO numrecs DO
      BEGIN
        rptr^.code := copy(linebuf,(i - 1) * 16 + 1,16) ;
        oldptr := rptr ;
        new(rptr) ;
        rptr^.code := '' ;
        rptr^.next := NIL ;
        oldptr^.next := rptr ;
      END ;
    IF tail > 0 THEN
      BEGIN
        oldptr := rptr ;
        new(rptr) ;
        rptr^.code := '' ;
        rptr^.next := NIL ;
        oldptr^.next := rptr ;
        rptr^.code := copy(linebuf,numrecs * 16 + 1,tail) ;
      END ;
    rptr^.next := NIL ;
    buftolist := lptr ;
  END ;
                                           (*  buftolist  *)

PROCEDURE listobuf(lptr:recptr) ;
      (*  converts linked list of 16-byte records
          to 80-byte line string (linebuf)  *)

  VAR
    rptr : recptr ;

  BEGIN                                    (*  listobuf  *)
    rptr := lptr ;
    linebuf := '' ;
    WHILE rptr <> NIL DO
      BEGIN
        linebuf := concat(linebuf,rptr^.code) ;
        rptr := rptr^.next ;
      END ;
    db('END LISTOBUF') ;
  END ;
                                           (* listobuf  *)

PROCEDURE writebuf ;
```

```
  VAR
    i : integer ;

  BEGIN                                     (*  writebuf  *)
    clearscreen(1,24) ;
    gotoxy(1,1) ;
    FOR i := 1 TO lastline DO
      BEGIN
        listobuf(bufarray[i]) ;
        writeln(linebuf) ;
      END ;
  END ;   (*  writebuf  *)

PROCEDURE scrolldn(rowtop,rowbot:integer) ;

  BEGIN
    regs.ax := 7 * 256 + 1 ; (* AH = 7; AL = 1 line blanked *)
    regs.cx := (rowtop - 1) * 256 ;
              (*  CH = row, CL = 0 : upper left corner  *)
    regs.dx := (rowbot - 1) * 256 + 79 ;
              (*  DH = row, DL = 80 : lower right corner  *)
    regs.bx := 7 * 256 ; (* black/white attribute *)
    intr($10,regs) ;
  END ;

PROCEDURE scrollup(rowtop,rowbot:integer) ;

  BEGIN
    regs.ax := 6 * 256 + 1 ; (* AH = 6;  AL = 1 line blanked *)
    regs.cx := (rowtop - 1) * 256 ;
              (*  CH = row, CL = 0 : upper left corner  *)
    regs.dx := (rowbot - 1) * 256 + 79 ;
              (*  DH = row, DL = 80 : lower right corner  *)
    regs.bx := 7 * 256 ; (*  black/white attribute *)
    intr($10,regs) ;
  END ;

PROCEDURE creturn ;   (*  carriage return   *)

  BEGIN
    db('START CRETURN') ;
    IF (bufarray[linenum] <> NIL) AND ( NOT buffed) THEN
      listobuf(bufarray[linenum]) ;
              (*  convert linked list to linebuf  *)
    linebuf := linebuf + chr(13) ;
    ch := ' ' ;
    bufarray[linenum] := buftolist ;
              (*  convert linebuf to linked list  *)
    buffed := false ;
    IF lastline < linenum THEN
      lastline := linenum ;
    pauseline := linenum ;
    linenum := linenum + 1 ;
    gotoxy(53,25) ;
    linebuf := '' ;
    IF row < 24 THEN
      row := row + 1
    ELSE
      BEGIN
        topline := topline + 1 ;
        scrollup(1,24) ;
      END ;
    col := 1 ;
    (*  status;  *)
    gotoxy(col,row) ;
    inserton := true ;
    db('END CRETURN') ;
  END ;

PROCEDURE cursup ;

  BEGIN
    IF linenum > 1 THEN
      BEGIN
        IF buffed THEN                    (*  line is in linebuf  *)
          BEGIN
            buffed := false ;
            bufarray[linenum] := buftolist ;
          END ;
        IF row > 1 THEN
          row := row - 1 ;
        gotoxy(col,row) ;
        linenum := linenum - 1 ;
        pauseline := linenum - 1 ;
        IF pauseline <= compline THEN
          recompile := true ;
      END ;
  END ;

PROCEDURE cursdn ;

  BEGIN
    IF linenum < lastline THEN
      BEGIN
        IF buffed THEN                    (*  line is in linebuf  *)
          BEGIN
            buffed := false ;
            bufarray[linenum] := buftolist ;
          END ;
        row := row + 1 ;
        gotoxy(col,row) ;
        linenum := linenum + 1 ;
        pauseline := linenum - 1 ;
        IF pauseline <= compline THEN
          recompile := true ;
      END ;
  END ;

PROCEDURE curslt ;

  BEGIN
    col := col - 1 ;
    gotoxy(col,row) ;
  END ;

PROCEDURE cursrt ;

  BEGIN
    col := col + 1 ;
    gotoxy(col,row) ;
  END ;

PROCEDURE pageup ;

  VAR
    stopline,j : integer ;

  BEGIN
    db('START PAGEUP') ;
    IF buffed THEN
      BEGIN
        buffed := false ;
        bufarray[linenum] := buftolist ;
      END ;
    clearscreen(1,25) ;
    gotoxy(1,1) ;
    topline := topline - 24 ;
    IF topline < 1 THEN
      topline := 1 ;
    IF topline > lastline - 23 THEN
      stopline := lastline
    ELSE
      stopline := topline + 23 ;
    FOR j := topline TO stopline DO
      BEGIN
        listobuf(bufarray[j]) ;
        writeln(linebuf) ;
      END ;
    linenum := topline ;
    pauseline := linenum - 1 ;
    IF pauseline <= compline THEN
      recompile := true ;
    row := 1 ;
    col := 1 ;
    gotoxy(col,row) ;
    db('END PAGEUP') ;
  END ;

PROCEDURE pagedn ;

  VAR
    stopline,j : integer ;

  BEGIN
    IF buffed THEN
      BEGIN
        buffed := false ;
        bufarray[linenum] := buftolist ;
      END ;
    clearscreen(1,24) ;
    gotoxy(1,1) ;
    topline := topline + 24 ;
    IF topline > lastline - 23 THEN
      BEGIN
        topline := lastline - 23 ;
        stopline := lastline ;
      END
    ELSE
      stopline := topline + 23 ;
    IF topline < 1 THEN
```

```
      topline := 1 ;
    FOR j := topline TO stopline DO
      BEGIN
        listobuf(bufarray[j]) ;
        writeln(linebuf) ;
      END ;
    linenum := topline ;
    pauseline := linenum - 1 ;
    IF pauseline <= compline THEN
      recompile := true ;
    row := 1 ;
    col := 1 ;
    gotoxy(col,row) ;
  END ;

PROCEDURE pagecomp ;

  VAR
    stopline,j : integer ;

  BEGIN
    IF buffed THEN
      BEGIN
        buffed := false ;
        bufarray[linenum] := buftolist ;
      END ;
    clearscreen(1,24) ;
    gotoxy(1,1) ;
    topline := compline - 10 ;
    IF topline < 1 THEN
      topline := 1 ;
    FOR j := topline TO compline DO
      BEGIN
        listobuf(bufarray[j]) ;
        writeln(linebuf) ;
      END ;
    linenum := compline ;
    row := compline - topline + 1 ;
    col := cc ;
  END ;

PROCEDURE insrtog ;
  (*  toggle insert mode  *)

  BEGIN
    IF inserton THEN
      inserton := false
    ELSE
      inserton := true ;
  END ;

PROCEDURE delchar ;
  (*  delete character  *)

  BEGIN
    IF NOT buffed THEN
      BEGIN
        listobuf(bufarray[linenum]) ;
        buffed := true ;
      END ;
    delete(linebuf,col,1) ;
    gotoxy(1,row) ;
    writeln(linebuf) ;
    gotoxy(length(linebuf) + 1,row) ;
    outch(' ') ;
    gotoxy(col,row) ;
  END ;

PROCEDURE insline ;

  VAR
    j : integer ;
    rptr : recptr ;

  BEGIN
    FOR j := linelimit DOWNTO linenum + 1 DO
      bufarray[j] := bufarray[j - 1] ;
    lastline := lastline + 1 ;
    new(rptr) ;
    rptr^.code := chr(13) ;
    rptr^.next := NIL ;
    bufarray[linenum] := rptr ;
    scrolldn(row,24) ;
    col := 1 ;
    gotoxy(col,row) ;
  END ;

PROCEDURE deline ;
  (*  delete line  *)

  VAR
    j : integer ;

  BEGIN
    (*  disposeline(linenum)  *)
    FOR j := linenum TO lastline - 1 DO
      bufarray[j] := bufarray[j + 1] ;
    bufarray[lastline] := NIL ;
    lastline := lastline - 1 ;
    scrollup(row,24) ;
    gotoxy(1,24) ;
    listobuf(bufarray[topline + 23]) ;
    writeln(linebuf) ;
    col := 1 ;
    gotoxy(col,row) ;
  END ;

PROCEDURE filesrch ;
(*  search directory for file name in command line parameter  *)

  VAR
    i,j,al : integer ;                    (*  AL register  *)
    fvar : text ;
    lbuf : STRING [80] ;
    fname : STRING [16] ;

  BEGIN
    fname := '' ;
    FOR i := 1 TO 8 DO
      IF chr(mem[cseg:$5c + i - 1]) > ' ' THEN
```

```
        fname := fname + chr(mem[cseg:$5c + i - 1]) ;
    fname := fname + '.' ;
    FOR i := 9 TO 12 DO
      IF chr(mem[cseg:$5c + i - 1]) > ' ' THEN
        fname := fname + chr(mem[cseg:$5c + i - 1]) ;
    assign(fvar,fname) ;
    al := 1 ;
    regs.ax := $11 * 256 ;
    regs.dx := $5c ; (*  FCB (File Control Block) address   *)
    regs.ds := cseg ; (*  code segment register  *)
    intr($21,regs) ;
    al := regs.ax - $11 * 256 ;
    IF al = 0 THEN                        (*  file exists   *)
      BEGIN
        newfile := false ;
        reset(fvar) ;
        j := 1 ;
        lastline := 0 ;
        WHILE NOT eof(fvar) DO
          BEGIN
            readln(fvar,linebuf) ;
            bufarray[j] := buftolist ;
            j := j + 1 ;
            lastline := lastline + 1 ;
          END ;
        close(fvar) ;
      END
    ELSE
                                          (*  new file  *)
      BEGIN
        newfile := true ;
        lastline := 0 ;
      END ;
  END ;

PROCEDURE edinit ;
  (*  initialize editor  *)

  VAR
    j : integer ;

  BEGIN
    clearscreen(1,24) ;
    row := 1 ;
    col := 1 ;
    gotoxy(col,row) ;
    buffed := false ;
    topline := 1 ;
    linenum := 1 ;
    pageup ;
    inserton := true ;
    c := ' ' ;
    IF newfile THEN
      bufarray[1] := NIL ;
  END ;

PROCEDURE compile ;

  BEGIN
    db('START COMPILE') ;
```

```
      pauseline := lastline + 1 ;
      IF buffed THEN
        BEGIN
          buffed := false ;
          bufarray[linenum] := buftolist ;
        END ;
      db('END COMPILE') ;
    END ;

  PROCEDURE edit ;
(*  var
    c: char;  *)

    BEGIN
      c := inkey ;
      IF ord(c) >= 32 THEN
        BEGIN
          IF NOT buffed THEN
            BEGIN
              linenum := topline + row - 1 ;
              IF bufarray[linenum] <> NIL THEN
                listobuf(bufarray[linenum]) ;
              buffed := true ;
            END ;
          WHILE col > length(linebuf) + 1 DO
            linebuf := linebuf + ' ' ;
          IF inserton THEN
            BEGIN
              insert(c,linebuf,col) ;
              gotoxy(1,row) ;
              writeln(linebuf) ;
              col := col + 1 ;
              gotoxy(col,row) ;
            END
          ELSE  (*  not in insert mode  *)
            BEGIN
              linebuf[col] := c ;
              outch(c) ;
              col := col + 1 ;
              gotoxy(col,row) ;
            END ;
          IF linenum <= compline THEN
            recompile := true ;
          pauseline := linenum - 1 ;
(*      if pauseline <= compline then
        recompile := true;
*)
        END  (*  regular character  *)
      ELSE   (*  control character  *)
        CASE ord(c) OF
          11,5 :                              (*   ^E   *)
          cursup ;
          10,24 :                             (*   ^X   *)
          cursdn ;
          8,19 :                              (*   ^S   *)
          curslt ;
          12,4 :                              (*   ^D   *)
          cursrt ;
          18 :                                (*   ^R   *)
          pageup ;
          16 :                                 (*  ^P  *)
          pagedn ;
          22 :                                 (*  ^V  *)
          insrtog ;
          14 :                                 (*  ^N  *)
          insline ;
          7 :                                  (*  ^G  *)
          delchar ;
          25 :                                 (*  ^Y  *)
          deline ;
          13 :                                 (*  CR  *)
          creturn ;
          3 :                                  (*  ^C  *)
          compile ;
          ELSE
            ;
        END ;                                  (*  case  *)
      (*  status;  *)
      gotoxy(col,row) ;
    END ;



(***************************************************************)



                        (*  ERR3.PAS  *)

PROCEDURE errormsg ;

  VAR
    k : integer ;

  BEGIN
    msg[0] := 'undef id  ' ;
    msg[1] := 'multi def ' ;
    msg[2] := 'identifier' ;
    msg[3] := 'program   ' ;
    msg[4] := ')         ' ;
    msg[5] := ':         ' ;
    msg[6] := 'syntax    ' ;
    msg[7] := 'ident, var' ;
    msg[8] := 'of        ' ;
    msg[9] := '(         ' ;
    msg[10] := 'id, array ' ;
    msg[11] := '[         ' ;
    msg[12] := ']         ' ;
    msg[13] := '..        ' ;
    msg[14] := ';         ' ;
    msg[15] := 'func. type' ;
    msg[16] := '=         ' ;
    msg[17] := 'boolean   ' ;
    msg[18] := 'convar typ' ;
    msg[19] := 'type      ' ;
    msg[20] := 'prog.param' ;
    msg[21] := 'too big   ' ;
    msg[22] := '.         ' ;
    msg[23] := 'typ (case)' ;
    msg[24] := 'character ' ;
    msg[25] := 'const id  ' ;
    msg[26] := 'index type' ;
    msg[27] := 'indexbound' ;
    msg[28] := 'no array  ' ;
    msg[29] := 'type id   ' ;
    msg[30] := 'undef type' ;
    msg[31] := 'no record ' ;
    msg[32] := 'boole type' ;
    msg[33] := 'arith type' ;
    msg[34] := 'integ     ' ;
    msg[35] := 'types     ' ;
    msg[36] := 'param type' ;
    msg[37] := 'variab id ' ;
    msg[38] := 'mstring   ' ;
    msg[39] := 'no.of pars' ;
    msg[40] := 'type      ' ;
    msg[41] := 'type      ' ;
    msg[42] := 'real type ' ;
    msg[43] := 'integer   ' ;
    msg[44] := 'var,const ' ;
    msg[45] := 'var, proc ' ;
    msg[46] := 'types (:=)' ;
    msg[47] := 'typ (case)' ;
    msg[48] := 'type      ' ;
    msg[49] := 'store ovfl' ;
    msg[50] := 'constant  ' ;
    msg[51] := ':=        ' ;
    msg[52] := 'the       ' ;
    msg[53] := 'until     ' ;
    msg[54] := 'do        ' ;
    msg[55] := 'to downto ' ;
    msg[56] := 'begin     ' ;
    msg[57] := 'end       ' ;
    msg[58] := 'factor    ' ;
  END                                         (*errormsg*)
  ;


(****************************************************************)


                        (*  LEX4.PAS  *)


PROCEDURE makelline(lineno:integer) ;
  (*  convert linked list of 16-byte records to array lline  *)

  VAR
    j : integer ;
    rptr : recptr ;
    compbuf : STRING [80] ;

  BEGIN
```

```
    rptr := bufarray[lineno] ;
    compbuf := '' ;
    WHILE rptr <> NIL DO
      BEGIN
        compbuf := concat(compbuf,rptr^.code) ;
        rptr := rptr^.next ;
      END ;
    ll := length(compbuf) ;
    FOR j := 1 TO ll DO
      lline[j] := compbuf[j] ;
    db('END MAKELLINE') ;
  END ;

  PROCEDURE nextch ;

    BEGIN
      IF keypressed THEN
        edit ;
      IF cc = ll THEN
        BEGIN
(*
      if ord(ch) = 26 then
        begin
          writeln;
          writeln('program incomplete');
          errormsg;
        end;
*)
          ll := 0 ;
          cc := 0 ;
          IF compline < pauseline THEN
            BEGIN
              db('COMPLINE < PAUSELINE') ;
              compline := compline + 1 ;
              gotoxy(32,25) ;
              (*          write(compline);  *)
              makelline(compline) ;
                (*  convert linked list line to array lline  *)
            END ;
        END ;                               (*  cc = ll  *)
      IF ll = 0 THEN
        ch := ' '
      ELSE
        BEGIN
          cc := cc + 1 ;
          ch := lline[cc] ;
(*      gotoxy(65, 25);
      write(ch);
      gotoxy(75, 25);
      write(cc);  *)
          gotoxy(col,row) ;
        END ;
    END ;
                                            (*  nextch  *)

  PROCEDURE error(n:integer) ;

    BEGIN
      IF NOT errorstate THEN
        BEGIN
        errorstate := true ;
        pauseline := compline - 1 ;
        recompile := true ;
        clearscreen(1,25) ;
        topline := compline + 1 ;
        IF topline < 1 THEN
          topline := 1 ;
        pageup ;
        gotoxy(1,20) ;
        writeln('LINE: ',compline,'     ','ERROR: ',msg[n]) ;
        col := cc ;
        row := compline - topline + 1 ;
        gotoxy(col,row) ;
        linenum := compline ;
      END ;                                   (*  if  *)
    (* status;  *)
  END                                         (*error*)
  ;

PROCEDURE fatal(n:integer) ;

  VAR
    msg : ARRAY [1..7] OF alfa ;

  BEGIN
    writeln ;
    errormsg ;
    msg[1] := 'identifier' ;
    msg[2] := 'procedures' ;
    msg[3] := 'reals     ' ;
    msg[4] := 'arrays    ' ;
    msg[5] := 'levels    ' ;
    msg[6] := 'code      ' ;
    msg[7] := 'strings   ' ;
    writeln(' compiler table for ',msg[n],' is too small') ;
    {goto 99}  (* terminate compilation*)
  END                                         (*fatal*)
  ;

PROCEDURE insymbol ;  (*reads next symbol*)

  LABEL
    1,2,3,quit ;

  VAR
    i,j,k,e : integer ;

  PROCEDURE readscale ;

    VAR
      s,sign : integer ;

    BEGIN
      nextch ;
      sign := 1 ;
      s := 0 ;
      IF ch = '+' THEN
        nextch
      ELSE IF ch = '-' THEN
        BEGIN
```

```
            nextch ;
            sign := - 1
          END ;
        WHILE ch IN ['0'..'9'] DO
          BEGIN
            s := 10 * s + ord(ch) - ord('0') ;
            nextch
          END ;
        e := s * sign + e
      END                                       (*readscale*)
      ;

    PROCEDURE adjustscale ;

      VAR
        s : integer ;
        d,t : real ;

      BEGIN
        IF k + e > emax THEN
          error(21)
        ELSE IF k + e < emin THEN
          rnum := 0
        ELSE
          BEGIN
            s := abs(e) ;
            t := 1.0 ;
            d := 10.0 ;
            REPEAT
              WHILE NOT odd(s) DO
                BEGIN
                  s := s DIV 2 ;
                  d := sqr(d)
                END ;
              s := s - 1 ;
              t := d * t
            UNTIL s = 0 ;
            IF e >= 0 THEN
              rnum := rnum * t
            ELSE
              rnum := rnum / t
          END
      END                                       (*adjustscale*)
      ;
    BEGIN                                       (*insymbol*)
      IF recompile THEN
        GOTO quit ;
1:
      WHILE ch <= ' ' DO
        nextch ;
      IF ch IN ['a'..'z'] THEN
        BEGIN                                   (*word*)
          k := 0 ;
          id := '          ' ;
          REPEAT
            IF k < alng THEN
              BEGIN
                k := k + 1 ;
                id[k] := ch
              END ;
      nextch
    UNTIL NOT (ch IN ['a'..'z','0'..'9']) ;
    i := 1 ;
    j := nkw ;                        (*binary search*)
    REPEAT
      k := (i + j) DIV 2 ;
      IF id <= key[k] THEN
        j := k - 1 ;
      IF id >= key[k] THEN
        i := k + 1
    UNTIL i > j ;
    IF i - 1 > j THEN
      sy := ksy[k]
    ELSE
      sy := ident ;
  END
ELSE IF ch IN ['0'..'9'] THEN
  BEGIN                               (*number*)
    k := 0 ;
    inum := 0 ;
    sy := intcon ;
    REPEAT
      inum := inum * 10 + ord(ch) - ord('0') ;
      k := k + 1 ;
      nextch
    UNTIL NOT (ch IN ['0'..'9']) ;
    IF (k > kmax) OR (inum > nmax) THEN
      BEGIN
        error(21) ;
        inum := 0 ;
        k := 0
      END ;
    IF ch = '.' THEN
      BEGIN
        nextch ;
        IF ch = '.' THEN
          ch := ':'
        ELSE
          BEGIN
            sy := realcon ;
            rnum := inum ;
            e := 0 ;
            WHILE ch IN ['0'..'9'] DO
              BEGIN
                e := e - 1 ;
                rnum := 10.0 * rnum + (ord(ch) - ord('0')) ;
                nextch
              END ;
            IF ch = 'e' THEN
              readscale ;
            IF e <> 0 THEN
              adjustscale
          END
      END
    ELSE IF ch = 'e' THEN
      BEGIN
        sy := realcon ;
        rnum := inum ;
        e := 0 ;
        readscale ;
```

```
              IF e <> 0 THEN
                adjustscale
            END ;
        END
      ELSE
        CASE ch OF
          ':' : BEGIN
                  nextch ;
                  IF ch = '=' THEN
                    BEGIN
                      sy := becomes ;
                      nextch
                    END
                  ELSE
                    sy := colon
                END ;
          '<' : BEGIN
                  nextch ;
                  IF ch = '=' THEN
                    BEGIN
                      sy := leg ;
                      nextch
                    END
                  ELSE IF ch = '>' THEN
                    BEGIN
                      sy := neg ;
                      nextch
                    END
                  ELSE
                    sy := lss
                END ;
          '>' : BEGIN
                  nextch ;
                  IF ch = '=' THEN
                    BEGIN
                      sy := geg ;
                      nextch
                    END
                  ELSE
                    sy := gtr
                END ;
          '.' : BEGIN
                  nextch ;
                  IF ch = '.' THEN
                    BEGIN
                      sy := colon ;
                      nextch
                    END
                  ELSE
                    sy := period
                END ;
          '''' : BEGIN
                   k := 0 ;
2:

                   nextch ;
                   IF ch = '''' THEN
                     BEGIN
                       nextch ;
                       IF ch <> '''' THEN
                         GOTO 3
                     END ;
                   IF sx + k = smax THEN
                     fatal(7) ;
                   stab[sx + k] := ch ;
                   k := k + 1 ;
                   IF cc = 1 THEN
                     BEGIN                      (* nd of line*)
                       k := 0 ;
                     END
                   ELSE
                     GOTO 2 ;
3:
                   IF k = 1 THEN
                     BEGIN
                       sy := charcon ;
                       inum := ord(stab[sx])
                     END
                   ELSE IF k = 0 THEN
                     BEGIN
                       error(38) ;
                       sy := charcon ;
                       inum := 0
                     END
                   ELSE
                     BEGIN
                       sy := mstring ;
                       inum := sx ;
                       sleng := k ;
                       sx := sx + k
                     END
                 END ;
          '(' : BEGIN
                  nextch ;
                  IF ch <> '*' THEN
                    sy := lparent
                  ELSE
                    BEGIN                       (*comment*)
                      nextch ;
                      REPEAT
                        WHILE ch <> '*' DO
                          nextch ;
                        nextch
                      UNTIL ch = ')' ;
                      nextch ;
                      GOTO 1
                    END
                END ;
          '+','-','*','/',')','=',',','[',']','#','&',';' :
          BEGIN
            sy := sps[ch] ;
            nextch
          END ;
          '$','%','@','\','~','{','}','^' : BEGIN
                                              error(24) ;
                                              nextch ;
                                              GOTO 1
                                            END
        END ;
quit:
    END                                     (*insymbol*)
    ;


(******************************************************************)


                         (*  ENTR.PAS  *)


PROCEDURE enter(x0:alfa ;
                x1:object ;
                x2:types ;
                x3:integer) ;

  BEGIN
    t := t + 1 ;  (*enter standard identifier*)
    WITH tab[t] DO
      BEGIN
        name := x0 ;
        link := t - 1 ;
        obj := x1 ;
        typ := x2 ;
        ref := 0 ;
        normal := true ;
        lev := 0 ;
        adr := x3
      END
  END                                                (*enter*)
  ;

  PROCEDURE enterarray(tp:types ;
                       l,h:integer) ;

    BEGIN
      IF l > h THEN
        error(27) ;
      IF (abs(l) > xmax) OR (abs(h) > xmax) THEN
        BEGIN
          error(27) ;
          l := 0 ;
          h := 0 ;
        END ;
      IF a = amax THEN
        fatal(4)
      ELSE
        BEGIN
          a := a + 1 ;
          WITH atab[a] DO
            BEGIN
              inxtyp := tp ;
              low := l ;
              high := h
            END
        END
    END                                              (*enterarray*)
    ;
```

```
PROCEDURE enterblock ;

  BEGIN
    IF b = bmax THEN
      fatal(2)
    ELSE
      BEGIN
        b := b + 1 ;
        btab[b].last := 0 ;
        btab[b].lastpar := 0
      END
  END                                     (*enterblock*)
  ;

PROCEDURE enterreal(x:real) ;

  BEGIN
    IF c2 = c2max - 1 THEN
      fatal(3)
    ELSE
      BEGIN
        rconst[c2 + 1] := x ;
        c1 := 1 ;
        WHILE rconst[c2 + 1] <> x DO
          c1 := c1 + 1 ;
        IF c1 > c2 THEN
          c2 := c1
      END
  END                                     (*enterreal*)
  ;

PROCEDURE emit(fct:integer) ;

  BEGIN
    IF lc = cmax THEN
      fatal(6) ;
    code[lc].f := fct ;
    lc := lc + 1
  END                                     (*emit*)
  ;

PROCEDURE emit1(fct,b:integer) ;

  BEGIN
    IF lc = cmax THEN
      fatal(6) ;
    WITH code[lc] DO
      BEGIN
        code[lc].f := fct ;
        y := b
      END ;
    lc := lc + 1
  END                                     (*emit1*)
  ;

PROCEDURE emit2(fct,a,b:integer) ;

  BEGIN
    IF lc = cmax THEN
```

```
         fatal(6) ;
     WITH code[lc] DO
       BEGIN
         f := fct ;
         x := a ;
         y := b
       END ;
     lc := lc + 1
   END                                        (*emit2*)
   ;

 PROCEDURE printtables ;

   VAR
     i : integer ;
     o : order ;

   BEGIN
writeln('Oidentifiers     link  obj  typ  ref  nrm  lev  adr') ;
writeln('printtables t = ',t) ;
     FOR i := btab[1].last TO t DO
       WITH tab[i] DO
         writeln(i,' ',name,link:5,ord(obj):5,ord(typ):5,ref:5,
               ord(normal):5,lev:5,adr:5) ;
     writeln('Oblocks    last lpar psze vsze') ;
     FOR i := 1 TO b DO
       WITH btab[i] DO
         writeln(i,last:5,lastpar:5,psize:5,vsize:5) ;
     writeln('Oarrays    xtyp etyp eref  low high elsz size') ;
     FOR i := 1 TO a DO
       WITH atab[i] DO
         writeln(i,ord(inxtyp):5,ord(eltyp):5,elref:5,low:5,
               high:5,elsize:5,size:5) ;
     writeln('Ocode:') ;
     FOR i := 0 TO lc - 1 DO
       BEGIN
         IF i MOD 5 = 0 THEN
           BEGIN
             writeln ;
             write(i:5)
           END ;
         o := code[i] ;
         write(o.f:5) ;
         IF o.f < 31 THEN
           IF o.f < 4 THEN
             write(o.x:2,o.y:5)
           ELSE
             write(o.y:7)
         ELSE
           write('     ') ;
         write(',')
       END ;
     writeln
   END                                        (*printtables*)
   ;
```

```
(****************************************************************)


                       (*  BLOCK3.PAS  *)


PROCEDURE block(fsys:symset ;
                isfun:boolean ;
                level:integer) ;

  LABEL
    quit ;

  TYPE
    conrec = RECORD
               CASE tp : types OF
                 ints,chars,bools : (i:integer) ;
                 reals : (r:real)
             END ;

  VAR
    dx : integer ;  (*data allocation index*)
    prt : integer ; (*t-index of this procedure*)
    prb : integer ; (*b-index of this procedure*)
    x : integer ;

  PROCEDURE skip(fsys:symset ;
                 n:integer) ;

    LABEL
      quit ;

    BEGIN
      error(n) ;
      GOTO quit ;
      WHILE NOT (sy IN fsys) DO
        insymbol ;
quit:
    END                                         (*skip*)
    ;

  PROCEDURE test(s1,s2:symset ;
                 n:integer) ;

    LABEL
      quit ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      IF NOT (sy IN s1) THEN
        skip(s1 + s2,n) ;
quit:
    END                                         (*test*)
    ;

  PROCEDURE testsemicolon ;
```

```
    LABEL
      quit ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      IF sy = semicolon THEN
        insymbol
      ELSE
        BEGIN
          error(14) ;
          GOTO quit ;
          IF sy IN [comma,colon] THEN
            insymbol
        END ;
      test([ident] + blockbegsys,fsys,6) ;
quit:
    END  (*testsemicolon*)
    ;

  PROCEDURE enter(id:alfa ;
                  k:object) ;

    LABEL
      quit ;

    VAR
      j,l : integer ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      IF t = tmax THEN
        fatal(1)
      ELSE
        BEGIN
          tab[0].name := id ;
          j := btab[display[level]].last ;
          l := j ;
          WHILE tab[j].name <> id DO
            BEGIN
              j := tab[j].link ;
              IF recompile THEN
                GOTO quit ;
            END ;
          IF j <> 0 THEN
            BEGIN
              error(1) ;
              GOTO quit ;
            END
          ELSE
            BEGIN
              t := t + 1 ;
              WITH tab[t] DO
                BEGIN
                  name := id ;
                  link := l ;
                  obj := k ;
                  typ := notyp ;
                  ref := 0 ;
                  lev := level ;
                  adr := 0
                END ;
              btab[display[level]].last := t
            END
        END ;
quit:
    END                                     (*enter*)
    ;

  FUNCTION loc(id:alfa) : integer ;

    LABEL
      quit ;

    VAR
      i,j : integer ;  (*locate id in table*)

    BEGIN
      IF recompile THEN
        GOTO quit ;
      i := level ;
      tab[0].name := id ;  (*sentinel*)
      REPEAT
        IF recompile THEN
          GOTO quit ;
        j := btab[display[i]].last ;
        WHILE tab[j].name <> id DO
          BEGIN
            j := tab[j].link ;
            IF recompile THEN
              GOTO quit ;
          END ;
        i := i - 1 ;
      UNTIL (i < 0) OR (j <> 0) ;
      IF j = 0 THEN
        BEGIN
          error(0) ;
          GOTO quit ;
        END ;
      loc := j ;
quit:
    END  (*loc*)
    ;

  PROCEDURE entervariable ;

    LABEL
      quit ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      IF sy = ident THEN
        BEGIN
          enter(id,variable) ;
          insymbol
        END
      ELSE
        error(2) ;
```

```
quit:
    END                                        (*entervariable*)
    ;

  PROCEDURE constant(fsys:symset ;
                     VAR c:conrec) ;

    LABEL
      quit ;

    VAR
      x,sign : integer ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      c.tp := notyp ;
      c.i := 0 ;
      test(constbegsys,fsys,50) ;
      IF sy IN constbegsys THEN
        BEGIN
          IF sy = charcon THEN
            BEGIN
              c.tp := chars ;
              c.i := inum ;
              insymbol
            END
          ELSE
            BEGIN
              sign := 1 ;
              IF sy IN [plus,minus] THEN
                BEGIN
                  IF sy = minus THEN
                    sign := - 1 ;
                  insymbol
                END ;
              IF sy = ident THEN
                BEGIN
                  x := loc(id) ;
                  IF recompile THEN
                    GOTO quit ;
                  IF x <> 0 THEN
                    IF tab[x].obj <> konstant THEN
                      BEGIN
                        error(25) ;
                        GOTO quit ;
                      END
                    ELSE
                      BEGIN
                        c.tp := tab[x].typ ;
                        IF c.tp = reals THEN
                          c.r := sign * rconst[tab[x].adr]
                        ELSE
                          c.i := sign * tab[x].adr
                      END ;
                  insymbol
                END
              ELSE IF sy = intcon THEN
                BEGIN
                  c.tp := ints ;
                  c.i := sign * inum ;
                  insymbol
                END
              ELSE IF sy = realcon THEN
                BEGIN
                  c.tp := reals ;
                  c.r := sign * rnum ;
                  insymbol
                END
              ELSE
                skip(fsys,50) ;
              IF recompile THEN
                GOTO quit ;
            END ;
          test(fsys,[],6)
        END ;
quit:
    END                                         (*constant*)
    ;

  PROCEDURE typ(fsys:symset ;
                VAR tp:types ;
                VAR rf,sz:integer) ;

    LABEL
      quit ;

    VAR
      x : integer ;
      eltp : types ;
      elrf : integer ;
      elsz,offset,t0,t1 : integer ;

    PROCEDURE arraytyp(VAR aref,arsz:integer) ;

      LABEL
        quit ;

      VAR
        eltp : types ;
        low,high : conrec ;
        elrd,elsz : integer ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        constant([colon,rbrack,rparent,ofsy] + fsys,low) ;
        IF low.tp = reals THEN
          BEGIN
            error(27) ;
            GOTO quit ;
            low.tp := ints ;
            low.i := 0
          END ;
        IF sy = colon THEN
          insymbol
        ELSE
          BEGIN
            error(13) ;
            GOTO quit ;
          END ;
        constant([rbrack,comma,rparent,ofsy] + fsys,high) ;
        IF high.tp <> low.tp THEN
          BEGIN
            error(27) ;
            GOTO quit ;
            high.i := low.i
          END ;
        enterarray(low.tp,low.i,high.i) ;
        aref := a ;
        IF sy = comma THEN
          BEGIN
            insymbol ;
            eltp := arrays ;
            arraytyp(elrf,elsz)
          END
        ELSE
          BEGIN
            IF sy = rbrack THEN
              insymbol
            ELSE
              BEGIN
                error(12) ;
                GOTO quit ;
                IF sy = rparent THEN
                  insymbol
              END ;
            IF sy = ofsy THEN
              insymbol
            ELSE
              BEGIN
                error(8) ;
                GOTO quit ;
              END ;
            typ(fsys,eltp,elrf,elsz)
          END ;
        WITH atab[aref] DO
          BEGIN
            arsz := (high - low + 1) * elsz ;
            size := arsz ;
            eltyp := eltp ;
            elref := elrf ;
            elsize := elsz
          END ;
quit:
      END                                     (*arraytyp*)
      ;
    BEGIN                                     (*typ*)
      IF recompile THEN
        GOTO quit ;
      tp := notyp ;
      rf := 0 ;
      sz := 0 ;
      test(typebegsys,fsys,10) ;
      IF sy IN typebegsys THEN
        BEGIN
          IF sy = ident THEN
            BEGIN
              x := loc(id) ;
              IF x <> 0 THEN
      WITH tab[x] DO
        IF obj <> type1 THEN
          BEGIN
            error(29) ;
            GOTO quit ;
          END
        ELSE
          BEGIN
            tp := typ ;
            rf := ref ;
            sz := adr ;
            IF tp = notyp THEN
              BEGIN
                error(30) ;
                GOTO quit ;
              END ;
          END ;
    insymbol
  END
ELSE IF sy = arraysy THEN
  BEGIN
    insymbol ;
    IF sy = lbrack THEN
      insymbol
    ELSE
      BEGIN
        error(11) ;
        GOTO quit ;
        IF sy = lparent THEN
          insymbol
      END ;
    tp := arrays ;
    arraytyp(rf,sz)
  END
ELSE
  BEGIN                             (*records*)
    insymbol ;
    enterblock ;
    tp := records ;
    rf := b ;
    IF level = lmax THEN
      fatal(5) ;
    level := level + 1 ;
    display[level] := b ;
    offset := 0 ;
    WHILE sy <> endsy DO
      BEGIN                         (*field section*)
        IF sy = ident THEN
          BEGIN
            t0 := t ;
            entervariable ;
            WHILE sy = comma DO
              BEGIN
                insymbol ;
                entervariable
              END ;
            IF sy = colon THEN
              insymbol
            ELSE
              BEGIN
```

```
                          error(5) ;
                          GOTO quit ;
                        END ;
                      tl := t ;
                      typ(fsys
                + [semicolon,endsy,comma,ident],eltp,elrf,elsz)
                           ;
                      WHILE t0 < tl DO
                        BEGIN
                          t0 := t0 + 1 ;
                          WITH tab[t0] DO
                            BEGIN
                              typ := eltp ;
                              ref := elrf ;
                              normal := true ;
                              adr := offset ;
                              offset := offset + elsz
                            END
                        END
                    END ;
                  IF sy <> endsy THEN
                    BEGIN
                      IF sy = semicolon THEN
                        insymbol
                      ELSE
                        BEGIN
                          error(14) ;
                          GOTO quit ;
                          IF sy = comma THEN
                            insymbol
                        END ;
                      test([ident,endsy,semicolon],fsys,6)
                    END
                END ;
              btab[rf].vsize := offset ;
              sz := offset ;
              btab[rf].psize := 0 ;
              insymbol ;
              level := level - 1
            END ;
          test(fsys,[],6)
        END ;
quit:
    END                                         (*typ*)
    ;

  PROCEDURE parameterlist ;  (*formal parameter list*)

    LABEL
      quit ;

    VAR
      tp : types ;
      rf,sz,x,t0 : integer ;
      valpar : boolean ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      insymbol ;
```

```
tp := notyp ;
rf := 0 ;
sz := 0 ;
test([ident,varsy],fsys + [rparent],7) ;
IF recompile THEN
  GOTO quit ;
WHILE sy IN [ident,varsy] DO
  BEGIN
    IF sy <> varsy THEN
      valpar := true
    ELSE
      BEGIN
        insymbol ;
        valpar := false
      END ;
    t0 := t ;
    entervariable ;
    WHILE sy = comma DO
      BEGIN
        insymbol ;
        entervariable ;
      END ;
    IF sy = colon THEN
      BEGIN
        insymbol ;
        IF sy <> ident THEN
          BEGIN
            error(2) ;
            GOTO quit ;
          END
        ELSE
          BEGIN
            x := loc(id) ;
            insymbol ;
            IF x <> 0 THEN
              WITH tab[x] DO
                IF obj <> typel THEN
                  BEGIN
                    error(29) ;
                    GOTO quit ;
                  END
                ELSE
                  BEGIN
                    tp := typ ;
                    rf := ref ;
                    IF valpar THEN
                      sz := adr
                    ELSE
                      sz := 1
                  END ;
          END ;
        test([semicolon,rparent],[comma,ident] + fsys,14)
      END
    ELSE
      BEGIN
        error(5) ;
        GOTO quit ;
      END ;
    WHILE t0 < t DO
      BEGIN
```

```
              t0 := t0 + 1 ;
              WITH tab[t0] DO
                BEGIN
                  typ := tp ;
                  ref := rf ;
                  normal := valpar ;
                  adr := dx ;
                  lev := level ;
                  dx := dx + sz
                END
            END ;
          IF sy <> rparent THEN
            BEGIN
              IF sy = semicolon THEN
                insymbol
              ELSE
                BEGIN
                  error(14) ;
                  GOTO quit ;
                  IF sy = comma THEN
                    insymbol
                END ;
              test([ident,varsy],[rparent] + fsys,6)
            END
        END                                 (*while*)
        ;
      IF sy = rparent THEN
        BEGIN
          insymbol ;
          test([semicolon,colon],fsys,6)
        END
      ELSE
        BEGIN
          error(4) ;
          GOTO quit ;
        END ;
quit:
    END                                     (*parameterlist*)
    ;

  PROCEDURE constantdeclaration ;

    LABEL
      quit ;

    VAR
      c : conrec ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      insymbol ;
      test([ident],blockbegsys,2) ;
      WHILE sy = ident DO
        BEGIN
          enter(id,konstant) ;
          insymbol ;
          IF sy = egl THEN
            insymbol
          ELSE
```

```
            BEGIN
              error(16) ;
              GOTO quit ;
              IF sy = becomes THEN
                insymbol
            END ;
          constant([semicolon,comma,ident] + fsys,c) ;
          tab[t].typ := c.tp ;
          tab[t].ref := 0 ;
          IF c.tp = reals THEN
            BEGIN
              enterreal(c.r) ;
              tab[t].adr := c1
            END
          ELSE
            tab[t].adr := c.i ;
          testsemicolon
        END ;
quit:    END  (*constantdeclaration*)
    ;

  PROCEDURE typedeclaration ;

    LABEL
      quit ;

    VAR
      tp : types ;
      rf,sz,t1 : integer ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      insymbol ;
      test([ident],blockbegsys,2) ;
      WHILE sy = ident DO
        BEGIN
          enter(id,typel) ;
          t1 := t ;
          insymbol ;
          IF sy = egl THEN
            insymbol
          ELSE
            BEGIN
              error(16) ;
              GOTO quit ;
              IF sy = becomes THEN
                insymbol
            END ;
          typ([semicolon,comma,ident] + fsys,tp,rf,sz) ;
          WITH tab[t1] DO
            BEGIN
              typ := tp ;
              ref := rf ;
              adr := sz
            END ;
          testsemicolon
        END ;
quit:
    END  (*typedeclaration*)
    ;

  PROCEDURE variabledeclaration ;

    LABEL
      quit ;

    VAR
      t0,t1,rf,sz : integer ;
      tp : types ;

    BEGIN
      IF recompile THEN
        GOTO quit ;
      insymbol ;
      WHILE sy = ident DO
        BEGIN
          t0 := t ;
          entervariable ;
          WHILE sy = comma DO
            BEGIN
              insymbol ;
              entervariable ;
            END ;
          IF sy = colon THEN
            insymbol
          ELSE
            BEGIN
              error(5) ;
              GOTO quit ;
            END ;
          t1 := t ;
          typ([semicolon,comma,ident] + fsys,tp,rf,sz) ;
          WHILE t0 < t1 DO
            BEGIN
              t0 := t0 + 1 ;
              WITH tab[t0] DO
                BEGIN
                  typ := tp ;
                  ref := rf ;
                  lev := level ;
                  adr := dx ;
                  normal := true ;
                  dx := dx + sz
                END
            END ;
          testsemicolon
        END ;
quit:
    END  (*variabledeclaration*)
    ;

  PROCEDURE procdeclaration ;

    LABEL
      quit ;

    VAR
      isfun : boolean ;
    BEGIN
      IF recompile THEN
        GOTO quit ;
      isfun := sy = functionsy ;
      insymbol ;
      IF sy <> ident THEN
        BEGIN
          error(2) ;
          GOTO quit ;
        END ;
      IF isfun THEN
        enter(id,funktion)
      ELSE
        enter(id,prozedure) ;
      tab[t].normal := true ;
      insymbol ;
      block([semicolon] + fsys,isfun,level + 1) ;
      IF sy = semicolon THEN
        insymbol
      ELSE
        BEGIN
          error(14) ;
          GOTO quit ;
        END ;
      emit(32 + ord(isfun))                 (*exit*)
      ;
quit:
    END  (*proceduredeclaration*)
    ;

  PROCEDURE statement(fsys:symset) ;

    LABEL
      quit ;

    VAR
      i : integer ;
      x : item ;

    PROCEDURE expression(fsys:symset ;
                         VAR x:item) ;
      FORWARD ;

    PROCEDURE selector(fsys:symset ;
                       VAR v:item) ;

      LABEL
        quit ;

      VAR
        x : item ;
        a,j : integer ;

      BEGIN  (*sy in [lparent, lbrack, period]*)
        IF recompile THEN
          GOTO quit ;
        REPEAT
          IF recompile THEN
            GOTO quit ;
          IF sy = period THEN
  BEGIN
    insymbol ;  (*field selector*)
    IF sy <> ident THEN
      BEGIN
        error(2) ;
        GOTO quit ;
      END
    ELSE
      BEGIN
        IF v.typ <> records THEN
          BEGIN
            error(31) ;
            GOTO quit ;
          END
        ELSE
          BEGIN  (*search field identifier*)
            j := btab[v.ref].last ;
            tab[0].name := id ;
            WHILE tab[j].name <> id DO
              BEGIN
                j := tab[j].link ;
                IF recompile THEN
                  GOTO quit ;
              END ;
            IF j = 0 THEN
              BEGIN
                error(0) ;
                GOTO quit ;
              END ;
            v.typ := tab[j].typ ;
            v.ref := tab[j].ref ;
            a := tab[j].adr ;
            IF a <> 0 THEN
              emit1(9,a) ;
            IF recompile THEN
              GOTO quit ;
          END ;
        insymbol
      END
  END
ELSE
  BEGIN   (*array selector*)
    IF sy <> lbrack THEN
      BEGIN
        error(11) ;
        GOTO quit ;
      END ;
    REPEAT
      IF recompile THEN
        GOTO quit ;
      insymbol ;
      expression(fsys + [comma,rbrack],x) ;
      IF v.typ <> arrays THEN
        BEGIN
          error(28) ;
          GOTO quit ;
        END
      ELSE
        BEGIN
          a := v.ref ;
```

```
                    IF atab[a].inxtyp <> x.typ THEN
                      BEGIN
                        error(26) ;
                        GOTO quit ;
                      END
                    ELSE IF atab[a].elsize = 1 THEN
                      emit1(20,a)
                    ELSE
                      emit1(21,a) ;
                    v.typ := atab[a].eltyp ;
                    v.ref := atab[a].elref ;
                    IF recompile THEN
                      GOTO quit ;
                  END
              UNTIL sy <> comma ;
              IF sy = rbrack THEN
                insymbol
              ELSE
                BEGIN
                  error(12) ;
                  GOTO quit ;
                  IF sy = rparent THEN
                    insymbol
                END
            END ;
          IF recompile THEN
            GOTO quit ;
        UNTIL NOT (sy IN [lbrack,lparent,period]) ;
        test(fsys,[],6) ;
quit:
      END                                   (*selector*)
      ;

    PROCEDURE call(fsys:symset ;
                   i:integer) ;

      LABEL
        quit ;

      VAR
        x : item ;
        lastp,cp,k : integer ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        emit1(18,i) ;                       (*mark stack*)
        lastp := btab[tab[i].ref].lastpar ;
        cp := i ;
        IF sy = lparent THEN
          BEGIN   (*actual parameter list*)
            REPEAT
              IF recompile THEN
                GOTO quit ;
              insymbol ;
              IF cp >= lastp THEN
                BEGIN
                  error(39) ;
                  GOTO quit ;
                END
```

```
ELSE
  BEGIN
    cp := cp + 1 ;
    IF tab[cp].normal THEN
      BEGIN                   (*value parameter*)
        expression(fsys + [comma,colon,rparent],x) ;
        IF x.typ = tab[cp].typ THEN
          BEGIN
            IF x.ref <> tab[cp].ref THEN
              BEGIN
                error(36) ;
                GOTO quit ;
              END
            ELSE IF x.typ = arrays THEN
              emit1(22,atab[x.ref].size)
            ELSE IF x.typ = records THEN
              emit1(22,btab[x.ref].vsize)
          END
        ELSE IF (x.typ = ints) AND
                  (tab[cp].typ = reals) THEN
          emit1(26,0)
        ELSE IF x.typ <> notyp THEN
          BEGIN
            error(36) ;
            GOTO quit ;
          END ;
      END
    ELSE
      BEGIN   (*variable parameter*)
        IF sy <> ident THEN
          BEGIN
            error(2) ;
            GOTO quit ;
          END
        ELSE
          BEGIN
            k := loc(id) ;
            insymbol ;
            IF k <> 0 THEN
              BEGIN
                IF tab[k].obj <> variable THEN
                  BEGIN
                    error(37) ;
                    GOTO quit ;
                  END ;
                x.typ := tab[k].typ ;
                x.ref := tab[k].ref ;
                IF tab[k].normal THEN
                  emit2(0,tab[k].lev,tab[k].adr)
                ELSE
                  emit2(1,tab[k].lev,tab[k].adr) ;
    IF sy IN [lbrack,lparent,period] THEN
      selector(fsys + [comma,colon,rparent],x) ;
    IF (x.typ <> tab[cp].typ) OR (x.ref <> tab[cp].
      ref) THEN
                  BEGIN
                    error(36) ;
                    GOTO quit ;
                  END ;
              END
```

```
                        END
                    END
                END ;
              test([comma,rparent],fsys,6) ;
              IF recompile THEN
                GOTO quit ;
            UNTIL sy <> comma ;
            IF sy = rparent THEN
              insymbol
            ELSE
              BEGIN
                error(4) ;
                GOTO quit ;
              END ;
          END ;
        IF cp < lastp THEN
          BEGIN
            error(39) ;
            GOTO quit ;
          END ;  (*too few actual parameters*)
        emit1(19,btab[tab[i].ref].psize - 1) ;
        IF tab[i].lev < level THEN
          emit2(3,tab[i].lev,level) ;
quit:
      END                                   (*call*)
      ;

    FUNCTION resulttype(a,b:types) : types ;

      LABEL
        quit ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        IF (a > reals) OR (b > reals) THEN
          BEGIN
            error(33) ;
            GOTO quit ;
            resulttype := notyp
          END
        ELSE IF (a = notyp) OR (b = notyp) THEN
          resulttype := notyp
        ELSE IF a = ints THEN
          IF b = ints THEN
            resulttype := ints
          ELSE
            BEGIN
              resulttype := reals ;
              emit1(26,1)
            END
        ELSE
          BEGIN
            resulttype := reals ;
            IF b = ints THEN
              emit1(26,0)
          END ;
quit:
      END                                   (*resulttype*)
      ;
```

```
PROCEDURE expression ;

  LABEL
    quit ;

  VAR
    y : item ;
    op : symbol ;

  PROCEDURE simpleexpression(fsys:symset ;
                             VAR x:item) ;

    LABEL
      quit ;

    VAR
      y : item ;
      op : symbol ;

    PROCEDURE term(fsys:symset ;
                   VAR x:item) ;

      LABEL
        quit ;

      VAR
        y : item ;
        op : symbol ;
        ts : typset ;

      PROCEDURE factor(fsys:symset ;
                       VAR x:item) ;

        LABEL
          quit ;

        VAR
          i,f : integer ;

        PROCEDURE standfct(n:integer) ;

          LABEL
            quit ;
            (* var ts : typset; *)

          BEGIN   (*standard function no. n*)
            IF recompile THEN
              GOTO quit ;
            IF sy = lparent THEN
              insymbol
            ELSE
              BEGIN
                error(9) ;
                GOTO quit ;
              END ;
            IF n < 17 THEN
              BEGIN
                expression(fsys + [rparent],x) ;
                CASE n OF
```

```
                      (*abs,sqr*)
                      0,2 : BEGIN
                              ts := [ints,reals] ;
                              tab[i].typ := x.typ ;
                              IF x.typ = reals THEN
                                n := n + 1
                            END ;
                      (*odd,chr*)
                      4,5 : ts := [ints] ;
                      (*ord*)
                      6 : ts := [ints,bools,chars] ;
                      (*succ,pred*)
                      7,8 : ts := [chars] ;
                      (*round,trunc*)
                      9,10,11,12,13,14,15,16 :
                      (*sin,cos,...*)
                      BEGIN
                        ts := [ints,reals] ;
                        IF x.typ = ints THEN
                          emit1(26,0)
                      END ;
                    END ;
                    IF x.typ IN ts THEN
                      emit1(8,n)
                    ELSE IF x.typ <> notyp THEN
                      BEGIN
                        error(48) ;
                        GOTO quit ;
                      END
                  END
                ELSE
                  (*eof,eoln*)
                  BEGIN                       (*n in [17,18]*)
                    IF sy <> ident THEN
                      BEGIN
                        error(2) ;
                        GOTO quit ;
                      END
                    ELSE IF id <> 'input     ' THEN
                      BEGIN
                        error(0) ;
                        GOTO quit ;
                      END
                    ELSE
                      insymbol ;
                    emit1(8,n) ;
                  END ;
                x.typ := tab[i].typ ;
                IF sy = rparent THEN
                  insymbol
                ELSE
                  BEGIN
                    error(4) ;
                    GOTO quit ;
                  END ;
quit:
              END                             (*standfct*)
              ;
            BEGIN                             (*factor*)
              IF recompile THEN
```

```
  GOTO quit ;
x.typ := notyp ;
x.ref := 0 ;
test(facbegsys,fsys,58) ;
IF recompile THEN
  GOTO quit ;
WHILE sy IN facbegsys DO
  BEGIN
    IF recompile THEN
      GOTO quit ;
    IF sy = ident THEN
      BEGIN
        i := loc(id) ;
        insymbol ;
        WITH tab[i] DO
          CASE obj OF
            konstant : BEGIN
                         x.typ := typ ;
                         x.ref := 0 ;
                         IF x.typ = reals THEN
                           emit1(25,adr)
                         ELSE
                           emit1(24,adr)
                       END ;
            variable : BEGIN
                         x.typ := typ ;
                         x.ref := ref ;
        IF sy IN [lbrack,lparent,period] THEN
                           BEGIN
                             IF normal THEN
                               f := 0
                             ELSE
                               f := 1 ;
                             emit2(f,lev,adr) ;
                             selector(fsys,x) ;
        IF x.typ IN stantyps THEN
                               emit(34)
                           END
                         ELSE
                           BEGIN
        IF x.typ IN stantyps THEN
                               IF normal THEN
                                 f := 1
                               ELSE
                                 f := 2
                             ELSE IF normal THEN
                               f := 0
                             ELSE
                               f := 1 ;
                             emit2(f,lev,adr)
                           END
                       END ;
            typel,prozedure : BEGIN
                                error(44) ;
                                GOTO quit ;
                              END ;
            funktion : BEGIN
                         x.typ := typ ;
                         IF lev <> 0 THEN
                           call(fsys,i)
```

```
                                       ELSE
                                         standfct(adr)
                                     END
                        END                 (*case,with*)
                    END
                  ELSE IF sy IN [charcon,intcon,realcon] THEN
                    BEGIN
                      IF sy = realcon THEN
                        BEGIN
                          x.typ := reals ;
                          enterreal(rnum) ;
                          emit1(25,c1)
                        END
                      ELSE
                        BEGIN
                          IF sy = charcon THEN
                            x.typ := chars
                          ELSE
                            x.typ := ints ;
                          emit1(24,inum)
                        END ;
                      x.ref := 0 ;
                      insymbol
                    END
                  ELSE IF sy = lparent THEN
                    BEGIN
                      insymbol ;
                      expression(fsys + [rparent],x) ;
                      IF sy = rparent THEN
                        insymbol
                      ELSE
                        BEGIN
                          error(4) ;
                          GOTO quit ;
                        END
                    END
                  ELSE IF sy = notsy THEN
                    BEGIN
                      insymbol ;
                      factor(fsys,x) ;
                      IF x.typ = bools THEN
                        emit(35)
                      ELSE IF x.typ <> notyp THEN
                        BEGIN
                          error(32) ;
                          GOTO quit ;
                        END
                    END ;
                  test(fsys,facbegsys,6)
                END ;                       (*while*)
              ;
quit:
            END                             (*factor*)
            ;
          BEGIN                             (*term*)
            ;
            IF recompile THEN
              GOTO quit ;
            factor(fsys + [times,rdiv,idiv,imod,andsy],x) ;
            WHILE sy IN [times,rdiv,idiv,imod,andsy] DO
```

```
BEGIN
  IF recompile THEN
    GOTO quit ;
  op := sy ;
  insymbol ;
  factor(fsys + [times,rdiv,idiv,imod,andsy],y) ;
  IF op = times THEN
    BEGIN
      x.typ := resulttype(x.typ,y.typ) ;
      CASE x.typ OF
        notyp : ;
        ints : emit(57) ;
        reals : emit(60) ;
      END
    END
  ELSE IF op = rdiv THEN
    BEGIN
     (*
  *)
      IF x.typ = ints THEN
        BEGIN
          emit1(26,1) ;
          x.typ := reals
        END ;
      IF y.typ = ints THEN
        BEGIN
          emit1(26,0) ;
          y.typ := reals
        END ;
      IF (x.typ = reals) AND (y.typ = reals) THEN
        emit(61)
      ELSE
        BEGIN
IF ((x.typ <> notyp) AND (y.typ <> notyp)) THEN
            BEGIN
              error(33) ;
              GOTO quit ;
            END ;
                                               (*
          *)
          x.typ := notyp
        END
    END
  ELSE IF op = andsy THEN
    BEGIN
IF ((x.typ = bools) AND (y.typ = bools)) THEN
        emit(56)
      ELSE
        BEGIN
IF ((x.typ <> notyp) AND (y.typ <> notyp)) THEN
            BEGIN
              error(32) ;
              GOTO quit ;
            END ;
          x.typ := notyp
        END
    END
  ELSE
    BEGIN   (*op in [indiv,imod]*)
      IF (x.typ = ints) AND (y.typ = ints) THEN
```

```
                      IF op = idiv THEN
                        emit(58)
                      ELSE
                        emit(59)
                    ELSE
                      BEGIN
                IF (x.typ <> notyp) AND (y.typ <> notyp) THEN
                          BEGIN
                            error(34) ;
                            GOTO quit ;
                          END ;
                        x.typ := notyp
                      END
                  END
              END ;
quit:
          END                                 (*term*)
          ;
        BEGIN   (*simpleexpression*)
          IF recompile THEN
            GOTO quit ;
          IF sy IN [plus,minus] THEN
            BEGIN
              op := sy ;
              insymbol ;
              term(fsys + [plus,minus],x) ;
              IF x.typ > reals THEN
                BEGIN
                  error(33) ;
                  GOTO quit ;
                END
              ELSE IF op = minus THEN
                emit(36)
            END
          ELSE
            term(fsys + [plus,minus,orsy],x) ;
          WHILE sy IN [plus,minus,orsy] DO
            BEGIN
              IF recompile THEN
                GOTO quit ;
              op := sy ;
              insymbol ;
              term(fsys + [plus,minus,orsy],y) ;
              IF op = orsy THEN
                BEGIN
                  IF (x.typ = bools) AND (y.typ = bools) THEN
                    emit(51)
                  ELSE
                    BEGIN
                IF (x.typ <> notyp) AND (y.typ <> notyp) THEN
                        BEGIN
                          error(32) ;
                          GOTO quit ;
                        END ;
                      x.typ := notyp
                    END
                END
              ELSE
                BEGIN
                  x.typ := resulttype(x.typ,y.typ) ;
                  CASE x.typ OF
                    notyp : ;
                    ints : IF op = plus THEN
                             emit(52)
                           ELSE
                             emit(53) ;
                    reals : IF op = plus THEN
                              emit(54)
                            ELSE
                              emit(55)
                  END
                END
            END ;
quit:
        END   (*simpleexpression*)
        ;
      BEGIN   (*expression*)
        IF recompile THEN
          GOTO quit ;
        simpleexpression(fsys + [egl,neg,lss,leg,gtr,geg],x) ;
        IF sy IN [egl,neg,lss,leg,gtr,geg] THEN
          BEGIN
            op := sy ;
            insymbol ;
            simpleexpression(fsys,y) ;
            IF (x.typ IN [notyp,ints,bools,chars])
                AND (x.typ = y.typ) THEN
              CASE op OF
                egl : emit(45) ;
                neg : emit(46) ;
                lss : emit(47) ;
                leg : emit(48) ;
                gtr : emit(49) ;
                geg : emit(50) ;
              END
            ELSE
              BEGIN
                IF x.typ = ints THEN
                  BEGIN
                    x.typ := reals ;
                    emit1(26,1)
                  END
                ELSE IF y.typ = ints THEN
                  BEGIN
                    y.typ := reals ;
                    emit1(26,0)
                  END ;
                IF (x.typ = reals) AND (y.typ = reals) THEN
                  CASE op OF
                    egl : emit(39) ;
                    neg : emit(40) ;
                    lss : emit(41) ;
                    leg : emit(42) ;
                    gtr : emit(43) ;
                    geg : emit(44) ;
                  END
                ELSE
                  BEGIN
                    error(35) ;
                    GOTO quit ;
```

```
         END
       END ;
      x.typ := bools
     END ;
quit:
   END                                    (*(expression*)
   ;

  PROCEDURE assignment(lv,ad:integer) ;

   LABEL
    quit ;

   VAR
    x,y : item ;
    f : integer ;
    (*tab[i].obj in [variable,prozedure]*)

   BEGIN
    IF recompile THEN
     GOTO quit ;
    x.typ := tab[i].typ ;
    x.ref := tab[i].ref ;
    IF tab[i].normal THEN
     f := 0
    ELSE
     f := 1 ;
    emit2(f,lv,ad) ;
    IF sy IN [lbrack,lparent,period] THEN
     selector([becomes,egl] + fsys,x) ;
    IF sy = becomes THEN
     insymbol
    ELSE
     BEGIN
      error(51) ;
      GOTO quit ;
      IF sy = egl THEN
       insymbol
     END ;
    expression(fsys,y) ;
    IF x.typ = y.typ THEN
     IF x.typ IN stantyps THEN
      emit(38)
     ELSE IF x.ref <> y.ref THEN
      BEGIN
       error(46) ;
       GOTO quit ;
      END
     ELSE IF x.typ = arrays THEN
      emit1(23,atab[x.ref].size)
     ELSE
      emit1(23,btab[x.ref].vsize)
    ELSE IF (x.typ = reals) AND (y.typ = ints) THEN
     BEGIN
      emit1(26,0) ;
      emit(38)
     END
    ELSE IF (x.typ <> notyp) AND (y.typ <> notyp) THEN
     BEGIN
      error(46) ;
```

```
            GOTO quit ;
          END ;
quit:
      END                                   (*assignment*)
      ;

    PROCEDURE compoundstatement ;

      LABEL
        quit ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        insymbol ;
        statement([semicolon,endsy] + fsys) ;
        WHILE sy IN [semicolon] + statbegsys DO
          BEGIN
            IF recompile THEN
              GOTO quit ;
            IF sy = semicolon THEN
              insymbol
            ELSE
              BEGIN
                error(14) ;
                GOTO quit ;
              END ;
            statement([semicolon,endsy] + fsys)
          END ;
        IF sy = endsy THEN
          insymbol
        ELSE
          BEGIN
            error(57) ;
            GOTO quit ;
          END ;
quit:
      END   (*compoundstatement*)
      ;

    PROCEDURE ifstatement ;

      LABEL
        quit ;

      VAR
        x : item ;
        lc1,lc2 : integer ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        insymbol ;
        expression(fsys + [thensy,dosy],x) ;
        IF NOT (x.typ IN [bools,notyp]) THEN
          BEGIN
            error(17) ;
            GOTO quit ;
          END ;
        lc1 := lc ;
        emit(11) ;                           (*jmpc*)
        IF sy = thensy THEN
          insymbol
        ELSE
          BEGIN
            error(52) ;
            GOTO quit ;
            IF sy = dosy THEN
              insymbol
          END ;
        statement(fsys + [elsesy]) ;
        IF sy = elsesy THEN
          BEGIN
            insymbol ;
            lc2 := lc ;
            emit(10) ;
            code[lc1].y := lc ;
            statement(fsys) ;
            code[lc2].y := lc
          END
        ELSE
          code[lc1].y := lc ;
quit:
      END                                    (*if statement*)
      ;

    PROCEDURE casestatement ;

      LABEL
        quit ;

      VAR
        x : item ;
        i,j,k,lc1 : integer ;
        casetab : ARRAY [1..csmax] OF PACKED RECORD
                                               val,lc : index
                                             END ;
        exittab : ARRAY [1..csmax] OF integer ;

      PROCEDURE caselabel ;

        LABEL
          quit ;

        VAR
          lab : conrec ;
          k : integer ;

        BEGIN
          IF recompile THEN
            GOTO quit ;
          constant(fsys + [comma,colon],lab) ;
          IF lab.tp <> x.typ THEN
            BEGIN
              error(47) ;
              GOTO quit ;
            END
          ELSE IF i = csmax THEN
            fatal(6)
          ELSE
```

```
            BEGIN
              i := i + 1 ;
              k := 0 ;
              casetab[i].val := lab.i ;
              casetab[i].lc := lc ;
              REPEAT
                IF recompile THEN
                  GOTO quit ;
                k := k + 1
              UNTIL casetab[k].val = lab.i ;
              IF k < i THEN
                BEGIN
                  error(1) ;
                  GOTO quit ;
                END   (*multiple definition*)
            END ;
quit:
        END   (*caselabel*)
        ;

      PROCEDURE onecase ;

        LABEL
          quit ;

        BEGIN
          IF recompile THEN
            GOTO quit ;
          IF sy IN constbegsys THEN
            BEGIN
              caselabel ;
              WHILE sy = comma DO
                BEGIN
                  IF recompile THEN
                    GOTO quit ;
                  insymbol ;
                  caselabel
                END ;
              IF sy = colon THEN
                insymbol
              ELSE
                BEGIN
                  error(5) ;
                  GOTO quit ;
                END ;
              statement([semicolon,endsy] + fsys) ;
              j := j + 1 ;
              exittab[j] := lc ;
              emit(10)
            END ;
quit:
        END                                 (*onecase*)
        ;
      BEGIN
        IF recompile THEN
          GOTO quit ;
        insymbol ;
        i := 0 ;
        j := 0 ;
        expression(fsys + [ofsy,comma,colon],x) ;
        IF NOT (x.typ IN [ints,bools,chars,notyp]) THEN
          BEGIN
            error(23) ;
            GOTO quit ;
          END ;
        lc1 := lc ;
        emit(12) ;                          (*jmpx*)
        IF sy = ofsy THEN
          insymbol
        ELSE
          BEGIN
            error(8) ;
            GOTO quit ;
          END ;
        onecase ;
        WHILE sy = semicolon DO
          BEGIN
            IF recompile THEN
              GOTO quit ;
            insymbol ;
            onecase
          END ;
        code[lc1].y := lc ;
        FOR k := 1 TO i DO
          BEGIN
            emit1(13,casetab[k].val) ;
            emit1(13,casetab[k].lc) ;
            IF recompile THEN
              GOTO quit ;
          END ;
        emit1(10,0) ;
        FOR k := 1 TO j DO
          code[exittab[k]].y := lc ;
        IF sy = endsy THEN
          insymbol
        ELSE
          BEGIN
            error(57) ;
            GOTO quit ;
          END ;
quit:
      END   (*casestatement*)
      ;

    PROCEDURE repeatstatement ;

      LABEL
        quit ;

      VAR
        x : item ;
        lc1 : integer ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        lc1 := lc ;
        insymbol ;
        statement([semicolon,untilsy] + fsys) ;
        WHILE sy IN [semicolon] + statbegsys DO
```

```
          BEGIN
            IF sy = semicolon THEN
              insymbol
            ELSE
              BEGIN
                error(14) ;
                GOTO quit ;
              END ;
            statement([semicolon,untilsy] + fsys)
          END ;
        IF sy = untilsy THEN
          BEGIN
            insymbol ;
            expression(fsys,x) ;
            IF NOT (x.typ IN [bools,notyp]) THEN
              BEGIN
                error(17) ;
                GOTO quit ;
              END ;
            emit1(11,lc1)
          END
        ELSE
          error(53) ;
quit:
      END   (*repeatstatement*)
      ;

    PROCEDURE whilestatement ;

      LABEL
        quit ;

      VAR
        x : item ;
        lc1,lc2 : integer ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        insymbol ;
        lc1 := lc ;
        expression(fsys + [dosy],x) ;
        IF NOT (x.typ IN [bools,notyp]) THEN
          BEGIN
            error(17) ;
            GOTO quit ;
          END ;
        lc2 := lc ;
        emit(11) ;
        IF sy = dosy THEN
          insymbol
        ELSE
          BEGIN
            error(54) ;
            GOTO quit ;
          END ;
        statement(fsys) ;
        emit1(10,lc1) ;
        code[lc2].y := lc ;
quit:
  END   (*whilestatement*)
  ;

PROCEDURE forstatement ;

  LABEL
    quit ;

  VAR
    cvt : types ;
    x : item ;
    i,f,lc1,lc2 : integer ;

  BEGIN
    IF recompile THEN
      GOTO quit ;
    insymbol ;
    IF sy = ident THEN
      BEGIN
        i := loc(id) ;
        insymbol ;
        IF i = 0 THEN
          cvt := ints
        ELSE IF tab[i].obj = variable THEN
          BEGIN
            cvt := tab[i].typ ;
            emit2(0,tab[i].lev,tab[i].adr) ;
            IF NOT (cvt IN [notyp,ints,bools,chars]) THEN
              BEGIN
                error(18) ;
                GOTO quit ;
              END
          END
        ELSE
          BEGIN
            error(37) ;
            GOTO quit ;
            cvt := ints
          END
      END
    ELSE
      skip([becomes,tosy,downtosy,dosy] + fsys,2) ;
    IF recompile THEN
      GOTO quit ;
    IF sy = becomes THEN
      BEGIN
        insymbol ;
        expression([tosy,downtosy,dosy] + fsys,x) ;
IF x.typ <> cvt THEN
          BEGIN
            error(19) ;
            GOTO quit ;
          END
      END
    ELSE
      skip([tosy,downtosy,dosy] + fsys,51) ;
    IF recompile THEN
      GOTO quit ;
    f := 14 ;
    IF sy IN [tosy,downtosy] THEN
```

```
          BEGIN
            IF sy = downtosy THEN
              f := 16 ;
            insymbol ;
            expression([dosy] + fsys,x) ;
            IF x.typ <> cvt THEN
              BEGIN
                error(19) ;
                GOTO quit ;
              END
          END
        ELSE
          skip([dosy] + fsys,55) ;
        IF recompile THEN
          GOTO quit ;
        lc1 := lc ;
        emit(f) ;
        IF sy = dosy THEN
          insymbol
        ELSE
          BEGIN
            error(54) ;
            GOTO quit ;
          END ;
        lc2 := lc ;
        statement(fsys) ;
        emit1(f + 1,lc2) ;
        code[lc1].y := lc ;
quit:
      END                                   (*forstatement*)
      ;

    PROCEDURE standproc(n:integer) ;

      LABEL
        quit ;

      VAR
        i,f : integer ;
        x,y : item ;

      BEGIN
        IF recompile THEN
          GOTO quit ;
        CASE n OF
          1,2 : BEGIN                       (*read*)
                  IF                        (* not *)
                    iflag THEN
                    BEGIN
                      error(20) ;
                      GOTO quit ;
                      iflag := true
                    END ;
                  IF sy = lparent THEN
                    BEGIN
                      REPEAT
                        IF recompile THEN
                          GOTO quit ;
                        insymbol ;
                        IF sy <> ident THEN
                BEGIN
                  error(2) ;
                  GOTO quit ;
                END
              ELSE
                BEGIN
                  i := loc(id) ;
                  insymbol ;
                  IF i <> 0 THEN
                    IF tab[i].obj <> variable THEN
                      BEGIN
                        error(37) ;
                        GOTO quit ;
                      END
                    ELSE
                      BEGIN
                        x.typ := tab[i].typ ;
                        x.ref := tab[i].ref ;
                        IF tab[i].normal THEN
                          f := 0
                        ELSE
                          f := 1 ;
                      emit2(f,tab[i].lev,tab[i].adr) ;
            IF sy IN [lbrack,lparent,period] THEN
                selector(fsys + [comma,rparent],x) ;
            IF x.typ IN [ints,reals,chars,notyp] THEN
                          emit1(27,ord(x.typ))
                        ELSE
                          BEGIN
                            error(40) ;
                            GOTO quit ;
                          END
                      END
                END ;
              test([comma,rparent],fsys,6) ;
              IF recompile THEN
                GOTO quit ;
            UNTIL sy <> comma ;
            IF sy = rparent THEN
              insymbol
            ELSE
              BEGIN
                error(4) ;
                GOTO quit ;
              END
          END ;
        IF n = 2 THEN
          emit(62)
      END ;
3,4 : BEGIN                       (*write*)
        IF sy = lparent THEN
          BEGIN
            REPEAT
              IF recompile THEN
                GOTO quit ;
              insymbol ;
              IF sy = mstring THEN
                BEGIN
                  emit1(24,sleng) ;
                  emit1(28,inum) ;
```

```
                            insymbol
                          END
                        ELSE
                          BEGIN
                    expression(fsys + [comma,colon,rparent],x) ;
                            IF NOT (x.typ IN stantyps) THEN
                              BEGIN
                                error(41) ;
                                GOTO quit ;
                              END ;
                            IF sy = colon THEN
                              BEGIN
                                insymbol ;
                    expression(fsys + [comma,colon,rparent],y) ;
                                IF y.typ <> ints THEN
                                  BEGIN
                                    error(43) ;
                                    GOTO quit ;
                                  END ;
                                IF sy = colon THEN
                                  BEGIN
                                    IF x.typ <> reals THEN
                                      BEGIN
                                        error(42) ;
                                        GOTO quit ;
                                      END ;
                                    insymbol ;
                            expression(fsys + [comma,rparent],y) ;
                                    IF y.typ <> ints THEN
                                      BEGIN
                                        error(43) ;
                                        GOTO quit ;
                                      END ;
                                    emit(37)
                                  END
                                ELSE
                                  emit1(30,ord(x.typ))
                              END
                            ELSE
                              emit1(29,ord(x.typ))
                          END ;
                        IF recompile THEN
                          GOTO quit ;
                      UNTIL sy <> comma ;
                      IF sy = rparent THEN
                        insymbol
                      ELSE
                        BEGIN
                          error(4) ;
                          GOTO quit ;
                        END
                    END ;
                  IF n = 4 THEN
                    emit(63)
                END ;
        END                                 (*case*)
        ;
quit:
      END                                   (*standproc*)
      ;
```

```
    BEGIN                                   (*statement*)
      IF recompile THEN
        GOTO quit ;
      IF sy IN statbegsys + [ident] THEN
        CASE sy OF
          ident : BEGIN
                    i := loc(id) ;
                    insymbol ;
                    IF i <> 0 THEN
                      CASE tab[i].obj OF
                        konstant,typel : BEGIN
                                           error(45) ;
                                           GOTO quit ;
                                         END ;
                        variable : BEGIN
                           assignment(tab[i].lev,tab[i].adr) ;
                                     IF recompile THEN
                                       GOTO quit ;
                                   END ;
                        prozedure : IF tab[i].lev <> 0 THEN
                                      call(fsys,i)
                                    ELSE
                                      standproc(tab[i].adr) ;
                        funktion :IF tab[i].ref = display[level]
                            THEN assignment(tab[i].lev + 1,0)
                                   ELSE
                                     BEGIN
                                       error(45) ;
                                       GOTO quit ;
                                     END
                      END
                  END ;
          beginsy : compoundstatement ;
          ifsy : ifstatement ;
          casesy : casestatement ;
          whilesy : whilestatement ;
          repeatsy : repeatstatement ;
          forsy : forstatement ;
        END ;
      test(fsys,[],14) ;
quit:
    END                                     (*statement*)
    ;

  BEGIN                                     (*block*)
    IF recompile THEN
      GOTO quit ;
    dx := 5 ;
    prt := t ;
    IF level > lmax THEN
      fatal(5) ;
    test([lparent,colon,semicolon],fsys,7) ;
    enterblock ;
    display[level] := b ;
    prb := b ;
    tab[prt].typ := notyp ;
    tab[prt].ref := prb ;
    IF sy = lparent THEN
      parameterlist ;
    btab[prb].lastpar := t ;
```

```
btab[prb].psize := dx ;
IF isfun THEN
  IF sy = colon THEN
    BEGIN
      insymbol ;                           (*function type*)
      IF sy = ident THEN
        BEGIN
          x := loc(id) ;
          insymbol ;
          IF x <> 0 THEN
            IF tab[x].obj <> type1 THEN
              BEGIN
                error(29) ;
                GOTO quit ;
              END
            ELSE IF tab[x].typ IN stantyps THEN
              tab[prt].typ := tab[x].typ
            ELSE
              BEGIN
                error(15) ;
                GOTO quit ;
              END
        END
      ELSE
        skip([semicolon] + fsys,2) ;
      IF recompile THEN
        GOTO quit ;
    END
  ELSE
    BEGIN
      error(5) ;
      GOTO quit ;
    END ;
IF sy = semicolon THEN
  insymbol
ELSE
  BEGIN
    error(14) ;
    GOTO quit ;
  END ;
REPEAT
  IF recompile THEN
    GOTO quit ;
  IF sy = constsy THEN
    constantdeclaration ;
  IF sy = typesy THEN
    typedeclaration ;
  IF sy = varsy THEN
    variabledeclaration ;
  btab[prb].vsize := dx ;
  WHILE sy IN [proceduresy,functionsy] DO
    procdeclaration ;
  test([beginsy],blockbegsys + statbegsys,56) ;
  IF recompile THEN
    GOTO quit ;
UNTIL sy IN statbegsys ;
tab[prt].adr := lc ;
insymbol ;
statement([semicolon,endsy] + fsys) ;
WHILE sy IN [semicolon] + statbegsys DO
```

```
      BEGIN
        IF recompile THEN
          GOTO quit ;
        IF sy = semicolon THEN
          insymbol
        ELSE
          BEGIN
            error(14) ;
            GOTO quit ;
          END ;
        statement([semicolon,endsy] + fsys)
      END ;
    IF sy = endsy THEN
      insymbol
    ELSE
      BEGIN
        error(57) ;
        GOTO quit ;
      END ;
    test(fsys + [period],[],6) ;
quit:
  END                                        (*block*)
  ;



(*****************************************************************)


                        (*  INTR.PAS  *)


PROCEDURE interpret ;
  (*global code, tab, btab*)

  VAR
    ir : order ;   (*instruction buffer*)
    pc : integer ; (*program counter*)
    ps : (run,fin,caschk,divchk,inxchk,stkchk,
                        linchk,lngchk,redchk) ;
    t : integer ;  (*top stack index*)
    b : integer ;  (*base index*)
    lncnt,ocnt,blkcnt,chrcnt : integer ;     (*counters*)
    h1,h2,h3,h4 : integer ;
    fld : ARRAY [1..4] OF integer ; (*defalt field widths*)
    display : ARRAY [1..lmax] OF integer ;
    s : ARRAY [1..stacksize] OF      (* blockmark:  *)
    RECORD
      CASE types OF   (*  s[b+0] = fct result     *)
        ints : (i:integer) ;  (*  s[b+1] = return adr     *)
        reals : (r:real) ;    (*  s[b+2] = static link    *)
        bools : (b:boolean) ; (*  s[b+3] = dynamic link   *)
        chars : (c:char)      (*  s[b+4] = table index    *)
    END ;

  BEGIN                                      (* interpret*)
    s[1].i := 0 ;
```

```
s[2].i := 0 ;
s[3].i := - 1 ;
s[4].i := btab[1].last ;
b := 0 ;
display[1] := 0 ;
t := btab[2].vsize - 1 ;
pc := tab[s[4].i].adr ;
ps := run ;
lncnt := 0 ;
ocnt := 0 ;
chrcnt := 0 ;
fld[1] := 10 ;
fld[2] := 22 ;
fld[3] := 10 ;
fld[4] := 1 ;
REPEAT
  ir := code[pc] ;
  pc := pc + 1 ;
  ocnt := ocnt + 1 ;
  CASE ir.f OF
    0 : BEGIN                           (*load address*)
          t := t + 1 ;
          IF t > stacksize THEN
            ps := stkchk
          ELSE
            s[t].i := display[ir.x] + ir.y
        END ;
    1 : BEGIN                           (*load value*)
          t := t + 1 ;
          IF t > stacksize THEN
            ps := stkchk
          ELSE
            s[t] := s[display[ir.x] + ir.y]
        END ;
    2 : BEGIN                           (*load indirect*)
          t := t + 1 ;
          IF t > stacksize THEN
            ps := stkchk
          ELSE
            s[t] := s[s[display[ir.x] + ir.y].i]
        END ;
    3 : BEGIN                           (*update display*)
          h1 := ir.y ;
          h2 := ir.x ;
          h3 := b ;
          REPEAT
            display[h1] := h3 ;
            h1 := h1 - 1 ;
            h3 := s[h3 + 2].i
          UNTIL h1 = h2
        END ;
    8 : CASE ir.y OF
          0 : s[t].i := abs(s[t].i) ;
          1 : s[t].r := abs(s[t].r) ;
          2 : s[t].i := sqr(s[t].i) ;
          3 : s[t].r := sqr(s[t].r) ;
          4 : s[t].b := odd(s[t].i) ;
          5 : BEGIN   (* s[t].c := chr(s[t].i) *)
                IF (s[t].i < 0) OR (s[t].i > 63) THEN
                  ps := inxchk
          END ;
      6 :      (* s[t].i := ord(s[t].c) *)
      ;
      7 : s[t].c := succ(s[t].c) ;
      8 : s[t].c := pred(s[t].c) ;
      9 : s[t].i := round(s[t].r) ;
      10 : s[t].i := trunc(s[t].r) ;
      11 : s[t].r := sin(s[t].r) ;
      12 : s[t].r := cos(s[t].r) ;
      13 : s[t].r := exp(s[t].r) ;
      14 : s[t].r := ln(s[t].r) ;
      15 : s[t].r := sqrt(s[t].r) ;
      16 : s[t].r := arctan(s[t].r) ;
      17 : BEGIN
             t := t + 1 ;
             IF t > stacksize THEN
               ps := stkchk
             ELSE
               s[t].b := eoln(input)
           END ;
      18 : BEGIN
             t := t + 1 ;
             IF t > stacksize THEN
               ps := stkchk
             ELSE
               s[t].b := eoln(input)
           END ;
    END ;
9 : s[t].i := s[t].i + ir.y ;       (*offset*)
10 : pc := ir.y ;                   (*jump*)
11 : BEGIN   (*conditional jump*)
       IF NOT s[t].b THEN
         pc := ir.y ;
       t := t - 1
     END ;
12 : BEGIN                          (*switch*)
       h1 := s[t].i ;
       t := t - 1 ;
       h2 := ir.y ;
       h3 := 0 ;
       REPEAT
         IF code[h2].f <> 13 THEN
           BEGIN
             h3 := 1 ;
             ps := caschk
           END
         ELSE IF code[h2].y = h1 THEN
           BEGIN
             h3 := 1 ;
             pc := code[h2 + 1].y
           END
         ELSE
           h2 := h2 + 2
       UNTIL h3 <> 0
     END ;
14 : BEGIN                          (*forlup*)
       h1 := s[t - 1].i ;
       IF h1 <= s[t].i THEN
         s[s[t - 2].i].i := h1
       ELSE
```

```
         BEGIN
           t := t - 3 ;
           pc := ir.y
         END
     END ;
15 : BEGIN                               (*for2up*)
       h2 := s[t - 2].i ;
       h1 := s[h2].i + 1 ;
       IF h1 <= s[t].i THEN
         BEGIN
           s[h2].i := h1 ;
           pc := ir.y
         END
       ELSE
         t := t - 3 ;
     END ;
16 : BEGIN                               (*for1down*)
       h1 := s[t - 1].i ;
       IF h1 >= s[t].i THEN
         s[s[t - 2].i].i := h1
       ELSE
         BEGIN
           pc := ir.y ;
           t := t - 3
         END
     END ;
17 : BEGIN                               (*for2down*)
       h2 := s[t - 2].i ;
       h1 := s[h2].i - 1 ;
       IF h1 >= s[t].i THEN
         BEGIN
           s[h2].i := h1 ;
           pc := ir.y
         END
       ELSE
         t := t - 3 ;
     END ;
18 : BEGIN                               (*mark stack*)
       h1 := btab[tab[ir.y].ref].vsize ;
       IF t + h1 > stacksize THEN
         ps := stkchk
       ELSE
         BEGIN
           t := t + 5 ;
           s[t - 1].i := h1 - 1 ;
           s[t].i := ir.y
         END
     END ;
19 : BEGIN                               (*call*)
       h1 := t - ir.y ;   (*h1 points to base*)
       h2 := s[h1 + 4].i ;(*h2 points to tab*)
       h3 := tab[h2].lev ;
       display[h3 + 1] := h1 ;
       h4 := s[h1 + 3].i + h1 ;
       s[h1 + 1].i := pc ;
       s[h1 + 2].i := display[h3] ;
       s[h1 + 3].i := b ;
       FOR h3 := t + 1 TO h4 DO
         s[h3].i := 0 ;
       b := h1 ;
       t := h4 ;
       pc := tab[h2].adr
     END ;
20 : BEGIN                              (*index1*)
       h1 := ir.y ;   (*h1 points to atab*)
       h2 := atab[h1].low ;
       h3 := s[t].i ;
       IF h3 < h2 THEN
         ps := inxchk
       ELSE IF h3 > atab[h1].high THEN
         ps := inxchk
       ELSE
         BEGIN
           t := t - 1 ;
           s[t].i := s[t].i + (h3 - h2)
         END
     END ;
21 : BEGIN                              (*index*)
       h1 := ir.y ;  (*h1 points to atab*)
       h2 := atab[h1].low ;
       h3 := s[t].i ;
       IF h3 < h2 THEN
         ps := inxchk
       ELSE IF h3 > atab[h1].high THEN
         ps := inxchk
       ELSE
         BEGIN
           t := t - 1 ;
         s[t].i := s[t].i + (h3 - h2) * atab[h1].elsize
         END
     END ;
22 : BEGIN                              (*load block*)
       h1 := s[t].i ;
       t := t - 1 ;
       h2 := ir.y + t ;
       IF h2 > stacksize THEN
         ps := stkchk
       ELSE
         WHILE t < h2 DO
           BEGIN
             t := t + 1 ;
             s[t] := s[h1] ;
             h1 := h1 + 1
           END
     END ;
23 : BEGIN                              (*copy block*)
       h1 := s[t - 1].i ;
       h2 := s[t].i ;
       h3 := h1 + ir.y ;
       WHILE h1 < h3 DO
         BEGIN
           s[h1] := s[h2] ;
           h1 := h1 + 1 ;
           h2 := h2 + 1
         END ;
       t := t - 2
     END ;
24 : BEGIN                              (*literal*)
       t := t + 1 ;
       IF t > stacksize THEN
```

```
           ps := stkchk
         ELSE
           s[t].i := ir.y
       END ;
  25 : BEGIN                          (*load real*)
         t := t + 1 ;
         IF t > stacksize THEN
           ps := stkchk
         ELSE
           s[t].r := rconst[ir.y]
       END ;
  26 : BEGIN
          (*float) h1 := t - ir.y; s[h1].i
    end ;
27: begin (*read*)
         IF eoln(input) THEN
           ps := redchk
         ELSE
           CASE ir.y OF
             1 : read(s[s[t].i].i) ;
             2 : read(s[s[t].i].r) ;
             4 : read(s[s[t].i].c) ;
           END ;
         t := t - 1
       END ;
  28 : BEGIN                          (*write string*)
         h1 := s[t].i ;
         h2 := ir.y ;
         t := t - 1 ;
         chrcnt := chrcnt + h1 ;
         IF chrcnt > lineleng THEN
           ps := lngchk ;
         REPEAT
           write(stab[h2]) ;
           h1 := h1 - 1 ;
           h2 := h2 + 1
         UNTIL h1 = 0
       END ;
  29 : BEGIN                          (*writel*)
         chrcnt := chrcnt + fld[ir.y] ;
         IF chrcnt > lineleng THEN
           ps := lngchk
         ELSE
           CASE ir.y OF
             1 : write(s[t].i:fld[1]) ;
             2 : write(s[t].r:fld[2]) ;
             3 : write(s[t].b:fld[3]) ;
             4 : write(s[t].c) ;
           END ;
         t := t - 1
       END ;
  30 : BEGIN                          (*writel*)
         chrcnt := chrcnt + s[t].i ;
         IF chrcnt > lineleng THEN
           ps := lngchk
         ELSE
           CASE ir.y OF
             1 : write(s[t - 1].i:s[t].i) ;
             2 : write(s[t - 1].r:s[t].i) ;
             3 : write(s[t - 1].b:s[t].i) ;
           4 : write(s[t - 1].c:s[t].i) ;
         END ;
       t := t - 2
     END ;
31 : ps := fin ;
32 : BEGIN   (*exit procedure*)
       t := b - 1 ;
       pc := s[b + 1].i ;
       b := s[b + 3].i
     END ;
33 : BEGIN   (*exit function*)
       t := b ;
       pc := s[t + 1].i ;
       b := s[b + 3].i
     END ;
34 : s[t] := s[s[t].i] ;
35 : s[t].b := NOT s[t].b ;
36 : s[t].i := - s[t].i ;
37 : BEGIN
       chrcnt := chrcnt + s[t - 1].i ;
       IF chrcnt > lineleng THEN
         ps := lngchk
       ELSE
         write(s[t - 2].r:s[t - 1].i:s[t].i) ;
       t := t - 3
     END ;
38 : BEGIN                          (*store*)
       s[s[t - 1].i] := s[t] ;
       t := t - 2
     END ;
39 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].r = s[t + 1].r
     END ;
40 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].r <> s[t + 1].r
     END ;
41 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].r < s[t + 1].r
     END ;
42 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].r <= s[t + 1].r
     END ;
43 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].r > s[t + 1].r
     END ;
44 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].r >= s[t + 1].r
     END ;
45 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].i = s[t + 1].i
     END ;
46 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].i <> s[t + 1].i
     END ;
47 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].i < s[t + 1].i
     END ;
48 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].i <= s[t + 1].i
     END ;
49 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].i > s[t + 1].i
     END ;
50 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].i >= s[t + 1].i
     END ;
51 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].b OR s[t + 1].b
     END ;
52 : BEGIN
       t := t - 1 ;
       s[t].i := s[t].i + s[t + 1].i
     END ;
53 : BEGIN
       t := t - 1 ;
       s[t].i := s[t].i - s[t + 1].i
     END ;
54 : BEGIN
       t := t - 1 ;
       s[t].r := s[t].r + s[t + 1].r ;
     END ;
55 : BEGIN
       t := t - 1 ;
       s[t].r := s[t].r - s[t + 1].r ;
     END ;
56 : BEGIN
       t := t - 1 ;
       s[t].b := s[t].b AND s[t + 1].b
     END ;
57 : BEGIN
       t := t - 1 ;
       s[t].i := s[t].i * s[t + 1].i
     END ;
58 : BEGIN
       t := t - 1 ;
       IF s[t + 1].i = 0 THEN
         ps := divchk
       ELSE
         s[t].i := s[t].i DIV s[t + 1].i
     END ;
59 : BEGIN
       t := t - 1 ;
       IF s[t + 1].i = 0 THEN
         ps := divchk
       ELSE
         s[t].i := s[t].i MOD s[t + 1].i
     END ;
```

```
    60 : BEGIN
           t := t - 1 ;
           s[t].r := s[t].r * s[t + 1].r ;
         END ;
    61 : BEGIN
           t := t - 1 ;
           s[t].r := s[t].r / s[t + 1].r ;
         END ;
    62 : IF eoln(input) THEN
           ps := redchk
         ELSE
           readln ;
    63 : BEGIN
           writeln ;
           lncnt := lncnt + 1 ;
           chrcnt := 0 ;
           IF lncnt > linelimit THEN
             ps := linchk
         END
  END                                     (*case*)
  ;
UNTIL ps <> run ;
IF ps <> fin THEN
  BEGIN
    writeln ;
    write('0halt at',pc:5,'because of') ;
    CASE ps OF
      caschk : writeln('undefined case') ;
      divchk : writeln('divison by 0') ;
      inxchk : writeln('invalid index') ;
      stkchk : writeln('storage overflow') ;
      linchk : writeln('too much output') ;
      lngchk : writeln('line too long') ;
      redchk : writeln('reading past end of file') ;
    END ;
    h1 := b ;
    blkcnt := 10 ;   (*post mortem dump*)
    REPEAT
      writeln ;
      blkcnt := blkcnt - 1 ;
      IF blkcnt = 0 THEN
        h1 := 0 ;
      h2 := s[h1 + 4].i ;
      IF h1 <> 0 THEN
      writeln(' ',tab[h2].name,'called at',s[h1 + 1].i:5) ;
      h2 := btab[tab[h2].ref].last ;
      WHILE h2 <> 0 DO
        WITH tab[h2] DO
          BEGIN
            IF obj = variable THEN
              IF typ IN stantyps THEN
                BEGIN
                  write('    ',name,' = ') ;
                  IF normal THEN
                    h3 := h1 + adr
                  ELSE
                    h3 := s[h1 + adr].i ;
                  CASE typ OF
                    ints : writeln(s[h3].i) ;
                    reals : writeln(s[h3].r) ;
```

```
                        bools : writeln(s[h3].b) ;
                        chars : writeln(s[h3].c) ;
                      END
                    END ;
                h2 := link
              END ;
          h1 := s[h1 + 3].i
        UNTIL h1 < 0 ;
      END ;
    writeln ;
    writeln(ocnt,'steps')
  END                                       (*interpret*)
  ;


(****************************************************************)
```